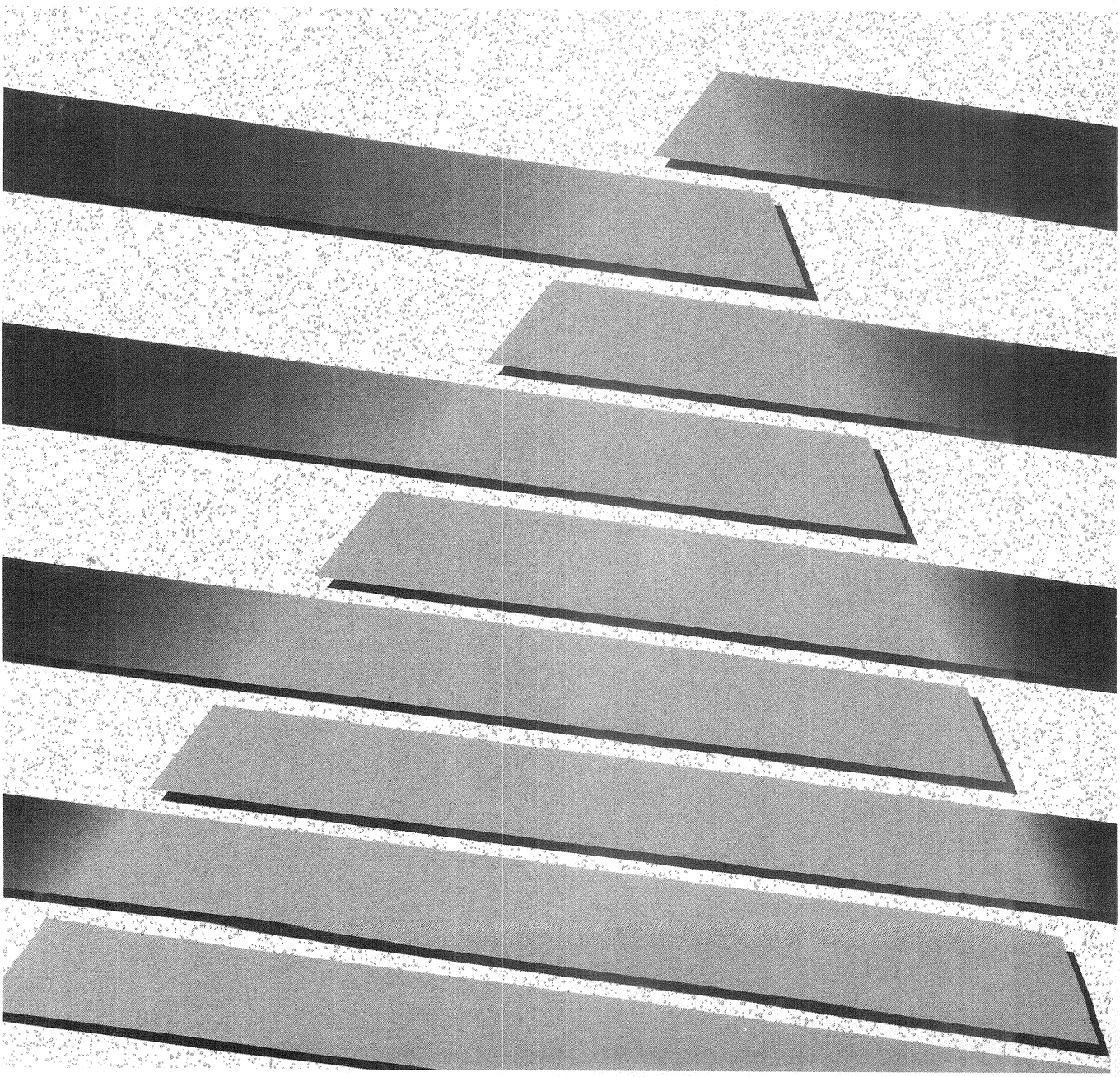
ALLEN-BRADLEY

PLC-5 Programming Software

Instruction Set Reference



Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI–1.1, "Safety Guidelines For The Application, Installation and Maintenance of Solid State Control" (available from your local Allen-Bradley office) describes some important differences between solid-state equipment and electromechanical devices which should be taken into consideration when applying products such as those described in this publication.

Reproduction of the contents of this copyrighted publication, in whole or in part, without written permission of Allen–Bradley Company, Inc. is prohibited.

Throughout this manual we make notes to alert you to possible injury to people or damage to equipment under specific circumstances.



WARNING: Tells readers where people may be hurt if procedures are not followed properly.



CAUTION: Tells readers where machinery may be damaged or economic loss can occur if procedures are not followed properly.

Warnings and Cautions:

- Identify a possible trouble spot.
- Tell what causes the trouble.
- Give the result of improper action.
- Tell the reader how to avoid trouble.

Important: We recommend you frequently backup your application programs on appropriate storage medium to avoid possible data loss.

© 1992 Allen–Bradley Company, Inc.

PLC-5 Instruction Set Alphabetical Listing

* Available with PLC-5/30, PLC-5/40 and PLC-5/60 processors only

For This Instruction:	See Chapter:
ABL	17 *
ACB	17 *
ACI	17 *
ACN	17 *
ACS	4 *
ADD	4
AEX	17 *
AFI	13
AHL	17 *
AIC	17 *
AND	5
ARD	17 *
ARL	17 *
ASC	17 *
ASN	4 *
ASR	17 *
ATN	4 *
AVE	4
AWA	17 *
AWT	17 *
BRK	13 *
BSL	11
BTD	7
BTR	15
BTW	15
CLR	4
CMP	3
COP	9
COS	4 *
CPT	4
CTD	2
CTU	2
DDT	10
DEG	6 *
DFA	18
DIV	4

For This Instruction:	See Chapter:
DTR	10
EOT	13
EQU	3
FAL	9
FBC	10
FFL	11
FFU	11
FLL	9
FOR	13 *
FRD	6
FSC	9
GEQ	3
GRT	3
IIN	1
IOT	1
JMP	13
JSR	13
LBL	13
LEQ	3
LES	3
LFL	11
LFU	11
LN	4 *
LOG	4 *
MCR	13
MEQ	3
MOV	7
MSG	16
MVM	7
MUL	4
NEG	4
NEQ	3
NOT	5
NXT	13 *
ONS	13
OR	5

For This Instruction:	See Chapter:
OSF	13 *
OSR	13 *
OTE	1
OTL	1
OTU	1
PID	14
RAD	6 *
RES	2
RET	13
RTO	2
SBR	13
SDS	18
SFR	13 *
SIN	4 *
SQI	12
SQL	12
SQO	12
SQR	4
SRT	4 *
STD	4 *
SUB	4
TAN	4 *
TND	13
TOD	6
TOF	2
TON	2
UID	13 *
UIE	13 *
XIC	1
XIO	1
XPY	4 *
XOR	5

Selecting the Appropriate Instruction Category

See Table A for guidelines on choosing the appropriate instruction for the operation you want to perform. Table B lists some examples.

Table A
Choosing an Instruction Category

If You Want To Perform This Operation:	Use This Instruction Category:	
examine, check or control	2-state device or condition multiple 2-state devices or conditions	bit level multi-bit
move, copy, change, compute, compare	analog values, codes multiple sets of values	element level file instructions
convert	conversion instructions	
time or delay	timer	
count	counter	
shift or track	bit shift	
sequence	sequencer	
PID	PID	
message sending/receiving	message	
transfer data between modules	block transfer	
diagnostics, fault handling	diagnostics	
control the flow of your program	program control	

Table B
Example Operations

If Your Application Calls For:	Use:
detecting when a limit switch closes	bit-level
changing the temperature preset	element-level
transfer analog data	block transfer
turn on a motor 10 seconds after a pump is activated	timing
move 1 of 3 recipes into a work area	multi-element
keep track of parts as they move from station to station	shifting
keep track of total parts in a bin	counting

Summary of Changes

New Information Added To This Module

This list below summarizes the changes that have been made to the "Instruction Set Reference" module since the last printing:

For This New Information:	See Chapter:
Information about determining the length of an expression in a compare instruction	3 - Compare Instructions
Information about determining the length of an expression in a compute instruction	4 - Compute Instructions
The SFR instruction has been enhanced for PLc-5/30 series A and PLC-5/40, -5/60 series B processors	13 - Program Control Instructions
The MSG instruction has been enhanced for co-processor ASCII messages	16 - Message Instruction
PLC-5/40 and -5/60 series B processor instruction timing information has been added	Appendix A - Instruction Timing
PLC-5/30, -5/40 and -5/60 processors need an additional 2 words of memory in the file header	Appendix B - SFC Reference

To help you find new information and updated information in this release of the manual, we have included change bars as shown to the left of this paragraph.

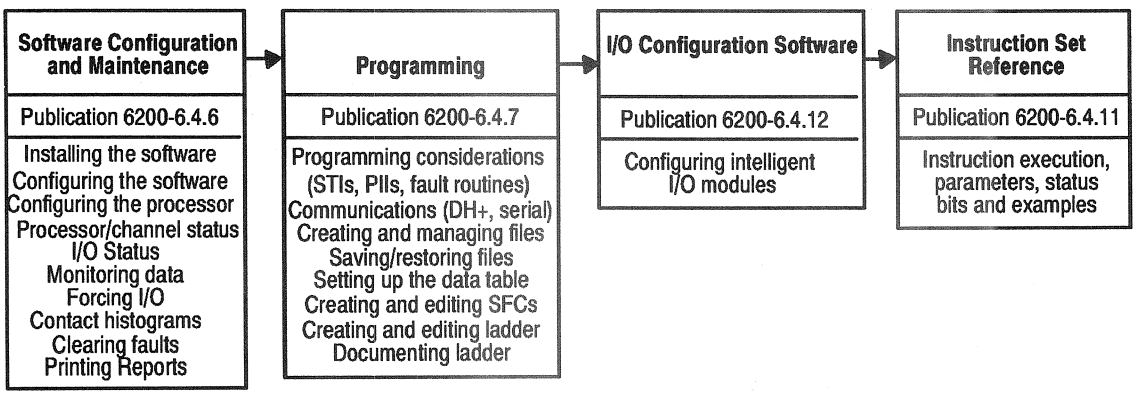
6200 Series Programming Software

6200 Series Programming Software Documentation

Your 6200 Series Programming Software documentation is organized into modules according to the tasks you perform. Each module represents a major programming task. This organization lets you easily find the information you want without reading through information that is not related to your current task. The documentation set contains these modules:

This Module:	Explains:
Software Configuration and Maintenance	how to install, start up, configure and maintain 6200 Series Software
Programming	considerations for programming your system, basics of file handling, and ladder and SFC program entry, edit and documentation
I/O Configuration	how to document, configure and edit intelligent I/O modules using I/O Configuration Software
Instruction Set Reference	instruction execution, parameters, status bits, and programming examples

Figure 1
6200 Series Programming Software Documentation Set



Where to Start

To use 6200 Series Software, you need a working knowledge of your Allen-Bradley PLC-5 processor.

If you are a new user, use the hardware installation manual (1785-6.6.1) to make sure your PLC-5 system is correctly installed. Then, start with the "Configuration and Maintenance" module to learn how to get your software running. Follow the arrows in Figure 1 to plan, create, and test your programs.

If you are already familiar with 6200 Series Software, you may only need the "Instruction Set Reference."

Conventions

This manual uses the following conventions to describe how to enter information:

- Words in square brackets represent actual keys that you press. For example: **[Enter]** or **[F1] - Online Programming/Documentation**
- Words that describe information that you have to provide are shown in italics. For example, if you have to type a file name, this is shown as: *filename*
- Messages and prompts that the terminal displays are shown as: `Press a function key`

Using the Index

At the back of each software module, there is a master index for the complete software documentation set. The page reference for the item you are looking for appears in the following format:

data file
  addressing: *Prog/Files* 4-1
  memory requirements: *Prog/Files* 5-3

where the name of the module that the entry can be found in is in *italics*, followed by the page reference.

For example, the entry for data file addressing (*Prog/Files*): is in the "Working with Files" section of the Programming module. Likewise, an entry with *Prog/SFC* is found in the "Entering and Editing SFCs" section of the Programming module. You would go to that section in the Programming module, and then find the page number.

Relay-Type Instructions XIC, XIO, OTE, OTL, OTU, IIN, IOT

Chapter 1
Using Relay-Type Instructions 1-1
I/O Image Files in Data Storage 1-1
Examine On (XIC) 1-2
Examine Off (XIO) 1-3
Output Energize (OTE) 1-3
Output Latch (OTL) 1-4
Output Unlatch (OTU) 1-5
Immediate Input (IIN) 1-5
Immediate Output (IOT) 1-6

Timer Instructions TON, TOF, RTO Counter Instructions CTU, CTD Reset RES

Chapter 2
Using Timers and Counters 2-1
Using Timers 2-1
Timer On Delay (TON) 2-3
Timer Off Delay (TOF) 2-6
Retentive Timer On (RTO) 2-8
Using Counters 2-11
Count Up (CTU) 2-12
Count Down (CTD) 2-14
Timer and Counter Reset (RES) 2-18

Compare Instructions CMP, EQU, GEQ, GRT, LEQ, LES, LIM, MEQ, NEQ

Chapter 3
Using Compare Instructions 3-1
Compare (CMP) 3-2
Equal to (EQU) 3-4
Greater than or Equal to (GEQ) 3-5
Greater than (GRT) 3-5
Less than or Equal to (LEQ) 3-5
Less than (LES) 3-6
Limit Test (LIM) 3-6
Mask Compare Equal to (MEQ) 3-7
Not Equal to (NEQ) 3-9

Compute Instructions CPT, ADD, AVE, CLR, DIV, MUL, NEG, SRT, SQR, STD, SUB

Chapter 4
Using Compute Instructions 4-1
Compute (CPT) 4-4
Arc Cosine (ACS) 4-10
Addition (ADD) 4-10
Arc Sine (ASN) 4-11
Arc Tangent (ATN) 4-12
Average (AVE) 4-12
Clear (CLR) 4-14
Cosine (COS) 4-15

Divide (DIV) 4-15
Natural Log (LN) 4-16
Log to the Base 10 (LOG) 4-17
Multiply (MUL) 4-17
Negate (NEG) 4-18
Sine (SIN) 4-18
Square Root (SQR) 4-19
Sort (SRT) 4-20
Standard Deviation (STD) 4-21
Subtract (SUB) 4-24
Tangent (TAN) 4-24
X to the Power of Y (XPY) 4-25

Logical Instructions AND, NOT, OR, XOR

Chapter 5
Using Logical Instructions 5-1
AND Operation (AND) 5-2
NOT Operation (NOT) 5-3
OR Operation (OR) 5-4
Exclusive OR Operation (XOR) 5-5

Conversion Instructions FRD and TOD

Chapter 6
Using the Conversion Instructions 6-1
Convert to BCD (TOD) 6-2
Convert from BCD (FRD) 6-2
Degree (DEG) 6-3
Radian (RAD) 6-4

Bit Modify and Move Instructions BTD, MOV, MVM

Chapter 7
Using Bit Modify and Move Instructions 7-1
Bit Distribute (BTD) 7-1
Move (MOV) 7-3
Masked Move (MVM) 7-3

File Instruction Concepts

Chapter 8
Concepts of File Operation 8-1
Entering Parameters 8-1
Using the Control Structure 8-2
Manipulating File Data 8-3
Choosing Modes of File Operation 8-4

File Instructions FAL, FSC, COP, FLL

Chapter 9

Using File Instructions 9-1
File Arithmetic and Logic (FAL) 9-1
FAL Copy Operations 9-3
FAL Arithmetic Operations 9-5
FAL Logic Operations 9-11
FAL Functions 9-12
File Search and Compare (FSC) 9-13
FSC Search and Compare Operations 9-15
File Copy (COP) 9-17
File Fill (FLL) 9-18

Diagnostic Instructions FBC, DDT, DTR

Chapter 10

Using Diagnostic Instructions 10-1
File Bit Comparison (FBC) and Diagnostic Detect (DDT) 10-1
Selecting the Search Mode 10-2
Entering Parameters 10-3
Using Status Bits 10-4
Data Transitional (DTR) 10-6

Shift Register Instructions BSL, BSR, FFL, FFU, LFL, LFU

Chapter 11

Applying Shift Registers 11-1
Using Bit Shift Instructions 11-1
Using FIFO and LIFO Instructions 11-5

Sequencer Instructions SQI, SQL, SQO

Chapter 12

Applying Sequencers 12-1
Using Sequencer Instructions 12-2

Program Control Instructions MCR, JMP, LBL, FOR, NXT, BRK, JSR, SBR, RET, TND, AFI, ONS, OSR, OSF, EOT

Chapter 13

Selecting Program Flow Instructions 13-1
Master Control Reset (MCR) 13-2
Jump (JMP) and Label (LBL) 13-3
For Next Loop (FOR, NXT), Break (BRK), and Return (RET) 13-5
Jump to Subroutine (JSR), Subroutine (SBR), and Return (RET) 13-8
Temporary End (TND) 13-14
Always False (AFI) 13-14
One Shot (ONS) 13-14
One Shot Rising (OSR) 13-15
One Shot Falling (OSF) 13-15
Sequential Function Chart Reset (SFR) 13-16
End of Transition (EOT) 13-17
User Interrupt Disable (UID) 13-17
User Interrupt Enable (UIE) 13-18

Process Control Instructions PID

Chapter 14

Using PID 14-1
Using PID Equations 14-2
Setting Input/Output Ranges 14-3
Implementing Scaling to Engineering Units 14-4
Setting the Dead Band 14-5
Selecting the Derivative Term 14-5
Setting Output Alarms 14-6
Using Output Limiting 14-6
Using Manual Mode Operation 14-6
Feedforward or Output Biasing 14-8
Resume Last State 14-8
PID Instruction 14-9
Using an Integer Data File Type for the Control Block 14-10
Using a PD File Type for the Control Block 14-14
Programming Considerations 14-21
Loop Considerations 14-22
Descaling Inputs 14-23
PID Examples 14-25

Block Transfer Instructions BTR and BTW

Chapter 15

Using Block Transfer Instructions 15-1
Block Transfer Read (BTR) and Block Transfer Write (BTW) 15-2
Entering Parameters 15-3
Using Status Bits 15-5
Using the Control Block 15-6
Selecting Continuous Operation 15-9
Selecting Non-Continuous Operation 15-10
Block Transfer Timing PLC-5/10, -5/12, -5/15, -5/25 15-12
Block Transfer Timing PLC-5/40, -5/60 15-13
Programming Examples 15-14

Message Instruction MSG

Chapter 16

Using the Message Instruction 16-1
Message (MSG) 16-1
Entering Parameters 16-1
Using Status Bits 16-5
Monitoring a Message Instruction 16-6
Selecting Continuous Operation 16-7
Selecting Non-Continuous Operation 16-8
MSG Timing 16-9
Error Codes 16-11

ASCII Instructions ABL, ACB, ACI, ACN, AEX, AIC, AHL, ARD, ARL, ASC, ASR, AWA, AWT PLC-5/40 and -5/60 Processors

Chapter 17
Using ASCII Instructions 17-1
Test Buffer for Line (ABL) 17-3
Number of Characters in Buffer (ACB) 17-4
ASCII String to Integer (ACI) 17-5
ASCII String Concatenate (ACN) 17-5
ASCII String Extract (AEX) 17-6
ASCII Set or Reset Handshake Lines (AHL) 17-7
ASCII Integer to String (AIC) 17-8
ASCII Read Characters (ARD) 17-8
ASCII Read Line (ARL) 17-10
ASCII String Search (ASC) 17-12
ASCII String Compare (ASR) 17-13
ASCII Write with Append (AWA) 17-13
ASCII Write (AWT) 17-15

Custom Application Routine Instructions SDS, DFA

Chapter 18
Chapter Objectives 18-1
Smart Directed Sequencer Overview 18-1
Programming the SDS Instruction 18-2
Diagnostic Fault Annunciator Overview 18-5
Programming the DFA Instruction 18-5

Instruction Timing and Memory Requirements

Appendix A
Instruction Timing and Memory Requirements A-1
Timing for PLC-5/30, -5/40 and -5/60 Processors A-2
Timing for PLc-5/10 -5/12, -5/15, -5/25 Processors A-9
Program Constants A-15
Direct and Indirect Elements PLC-5/30, -5/40, -5/60 Processors A-15
Direct and Indirect Elements for PLC-5/10, -5/12, -5/15, -5/25 ..... A-15

SFC Reference

Appendix B
Chapter Objectives B-1
SFC Status Information in the Processor Status File B-1
Memory Allocation B-3
Dynamic Constraints B-4
Scanning Sequences B-6
Run Times, PLC-5/10, -5/12, -5/15, -5/25 ..... B-12

Valid Data Types for Instruction Operands

Appendix C
Chapter Objectives C-1
Instruction Operands and Valid Data Types C-1

Chapter 1

Relay-Type Instructions XIC, XIO, OTE, OTL, OTU, IIN, IOT

Using Relay-Type Instructions

Use relay-type instructions to monitor and control the status of bits in the data table, such as input bits or timer control-word bits. The relay instructions include:

If You Want to:	Use this Instruction:	Found on Page:
Examine a bit for an ON condition	XIC	1-2
Examine a bit for an OFF condition	XIO	1-3
Hold a bit ON or OFF (non-retentive)	OTE	1-3
Latch a bit to ON (retentive)	OTL	1-4
Unlatch a bit to OFF (retentive)	OTU	1-5
Immediately update input image bits	IIN	1-5
Immediately update output image bits	IOT	1-6

With these instructions, you can address bits in all sections of data storage, but the examples in this chapter only show how to address bits in the I/O image files.

I/O Image Files in Data Storage

The input image file in the processor stores the status of input devices connected to input module terminals.

If the Input Device Is:	Then Its Corresponding Input Image Bit Is:
closed (on)	on (1)
open (off)	off (0)

You program instructions in ladder logic to monitor bits. Use a logical address for the bit.

The output image file controls the status of output devices wired to output module terminals.

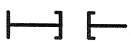
If the Output Image Bit Is:	Then Its Corresponding Output Device Image Is:
on (1)	energized (on)
off (0)	de-energized (off)

You program instructions in ladder logic to control bits.

Rung Logic

As each conditioning instruction is executed, the addressed bit is examined to see if it matches a certain condition (on or off). If a complete path of conditions examined for are found, the rung is set to true. The rung must contain a continuous path of true instructions from the start of the rung to the output for the output to be enabled.

Examine On (XIC)



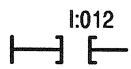
Description:
When a device closes its circuit, the module whose input terminal is wired to the device detects the closed circuit. The processor reflects this ON state in the data table. When the processor finds an XIC instruction that addresses the bit that corresponds to the input terminal, the processor determines whether the device is ON (closed). If the processor finds an ON state, it sets the logic for this instruction true; if the processor finds an OFF state, it sets the logic for the instruction false.

If the XIC instruction is the only conditioning instruction on the rung, the processor enables the output instruction when the XIC instruction is true (input closed). The processor disables an output instruction when the XIC instruction is false (input open).

The examine-on instruction is true or false depending on whether the processor finds an ON condition at the addressed bit.

If the Bit Is:	Then the Instruction Is:
on	true
off	false

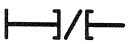
Example:



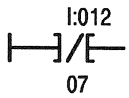
This instruction tells the processor: if you find an ON condition at bit I:012/07 in the input table, set this instruction true.

This bit corresponds to input terminal 7 of a module in I/O group 2 of I/O rack 1. If the input circuit is true, the instruction is true.

Examine Off (XIO)



Example:



This instruction tells the processor: if you find an OFF condition at bit I:012/07 in the input table, set this instruction true.

This bit corresponds to input terminal 7 of a module in I/O group 2 of I/O rack 1. If the input circuit is false, the instruction is true.

Description:

When a device opens its circuit, the modules whose input terminal is wired to the device detects an open circuit. The processor reflects this OFF state in the data table. When the processor finds an XIO instruction that addresses the bit that corresponds to the input terminal, the processor determines whether the device is OFF (open). If the processor finds an OFF state, it sets the logic for this instruction true. If the processor finds an ON state, it sets the XIO instruction to false.

If the XIO instruction is the only conditioning instruction on the rung, the processor enables the output instruction when the XIO instruction is true (input open).

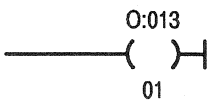
The examine off instruction is true or false depending on whether the processor finds an OFF condition at the addressed bit.

If the Bit Is:	Then the Instruction Is:
off	true
on	false

Energize (OTE)



Example:



This instruction tells the processor to turn ON bit O:013/01 of the output image table if the rung is true, and turn it OFF if the rung is false.

This bit corresponds to output terminal 1 of a module in I/O group 3 of I/O rack 1.

Description:

Use the OTE instruction to control a bit in memory. If the bit corresponds to an output module terminal, the device wired to this terminal is energized when the instruction is enabled and de-energized when the instruction is disabled. If the input conditions that precede the OTE instruction are true, the processor enables the OTE instruction. If the input conditions that precede the OTE instruction are false, the processor disables the OTE instruction. When rung conditions become false, the corresponding device de-energizes.

An OTE instruction is similar to a relay coil. The OTE instruction is controlled by preceding input instructions; the relay coil is controlled by contacts in its hard-wired rung.

The OTE instruction tells the processor to control the addressed bit based on the rung condition:

If the Rung Is:	Then the Processor Turns the Bit:
true	on
false	off

Latch (OTL)



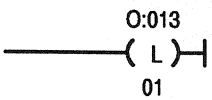
Description:
The OTL instruction is a retentive output instruction that can only turn on a bit (it cannot turn off a bit). This instruction is usually used in pairs with an OTU (unlatch) instruction, with both instructions addressing the same bit.

When you assign an address to an OTL instruction that corresponds to a terminal of an output module, the output device wired to this terminal is energized when the processor sets (enables) the bit in processor memory. If the input conditions that precede the OTL instruction are true, the processor enables the OTL instruction. When rung conditions become false (after being true), the bit remains set and the corresponding output device remains energized. Use the OTU instruction to turn OFF the bit you latched on with the OTL instruction.

When enabled, the latch instruction tells the processor to turn on the addressed bit. Thereafter, the bit remains on, regardless of the rung condition, until the bit is turned off, typically by an unlatch (OTU) instruction in another rung.

If the Rung Is:	Then the Processor Turns the Bit:
true	on
false	no change

Example:



This instruction tells the processor to turn ON output image table bit O:013/01 if the rung is true.

This bit corresponds to output terminal 1 of a module in I/O group 3 of I/O rack 1.

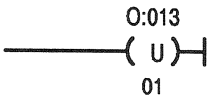
When the processor changes from Run to Program mode or when the processor loses power (and there is battery backup), the last true OTL instruction continues to control the bit in memory. The latched output device is energized even though the rung conditions that control the instruction may have gone false.

Important: The OTL instruction is retentive. When the processor loses power, is switched to Program mode or Test mode, or detects a major fault, outputs go off; but the states of retentive outputs are retained in memory. When the processor resumes operation in Run mode, retentive outputs immediately return to their previous states. Non-retentive outputs, such as OTE outputs, are reset.

Unlatch (OTU)



Example:



This instruction tells the processor to turn OFF output image table bit O:013/01 if the rung is true.

This bit corresponds to output terminal 1 of a module in I/O group 3 of I/O rack 1.

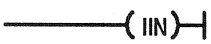
Description:
The OTU instruction is a retentive output instruction that can only turn off a bit (it cannot turn on a bit). This instruction is usually used in pairs with an OTL (output latch) instruction, with both instructions addressing the same bit. The OTU instruction turns OFF the bit, which was turned ON (latched) by the OTL instruction.

When the processor changes from Run to Program mode or when the processor loses power (and there is battery backup), the bit is retained in the state set by the last rung of the latch/unlatch pair that was true.

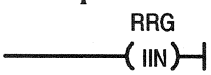
The unlatch instruction tells the processor to turn off the addressed bit based on the rung condition. Thereafter, the bit remains off, regardless of the rung condition, until it is turned on, typically by a OTL instruction in another rung.

If the rung is:	Then the processor turns the bit:
true	off
false	no change

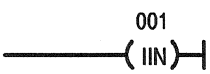
Immediate Input (IIN)



Example:



Where:
RR = I/O rack number
00-03 PLC-5/10, -5/12, -5/15
00-07 PLC-5/25, -5/30
00-17 PLC-5/40, -5/40L
00-27 PLC-5/60, -5/60L
G = I/O group number (0 - 7)



When the input conditions are true, the processor updates the input image word corresponding to I/O rack 0, group 1.

Description:
The immediate-input instruction is an output instruction that, when enabled, updates a word of input-image bits before the next normal input-image update.

For inputs in the **local chassis**, the program scan is interrupted while the inputs of the addressed I/O group are scanned. This sets the input-image bits to the current states of the inputs before the program scan continues. If the program reaches an enabled IIN instruction while a block-transfer with the local chassis is in progress, the processor completes the block-transfer before executing the IIN instruction.

For inputs in a **remote chassis**, the program scan is interrupted only to update the input image with the latest states of the inputs as found in the remote I/O buffer (from the most recent remote I/O scan). The inputs are not scanned before the program scan continues.

Place the rung with the IIN instruction immediately before rungs that examine critical input bits updated by the IIN instruction.

For the IIN instruction, you only need to enter the I/O rack number and the I/O group number; you do not enter a file number.

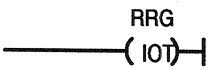
WARNING: Do not enter an address that includes a file number, such as I:027. The processor interprets the bit pattern found at that address as the I/O rack and I/O group number of the inputs to update. Unexpected operation will result with possible damage to equipment and injury to personnel.

For more information on I/O scanning and block transfers, see the block transfer instructions chapter in this module.

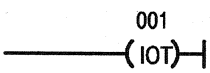
Immediate Output (IOT)



Example:



Where:
RR = I/O rack number
00-03 PLC-5/10, -5/12, -5/15
00-07 PLC-5/25, -5/30
00-17 PLC-5/40, -5/40L
00-27 PLC-5/60, -5/60L
G = I/O group number (0 - 7)



When the input conditions are true, the processor updates the outputs with the image word associated with I/O rack 0, group 1

Description:
The immediate-output instruction is an output instruction that, when enabled, updates an I/O group of outputs before the next normal output-image update.

For outputs in the **local chassis**, the program scan is interrupted while the outputs of the addressed I/O group are scanned. This sets the output circuits to the current states of the output bits in the output image table before the program scan continues. If the program reaches an enabled IOT instruction while a block-transfer is in progress, the processor completes the block-transfer before executing the IOT instruction.

For outputs in a **remote chassis**, the program scan is interrupted only to update the remote I/O buffer with the current states of the output-image bits. This makes these states immediately available for the next remote I/O scan while the program scan continues. The outputs are not scanned before the program scan continues.

Place the rung with the IOT output instruction immediately after rungs that control critical output image bits to be updated by the IOT instruction.

For the IOT instruction, you only need to enter the I/O rack number and the I/O group number; you do not need to enter the file number.

WARNING: Do not enter an address that includes a file number, such as O:027. The processor interprets the bit pattern found at that address as the I/O rack and I/O group number of the outputs to be updated. Unexpected operation will result with possible damage to equipment and injury to personnel.

For more information on I/O scanning and block transfers, see the block transfer instructions chapter in this module.

Chapter 2

Timer Instructions TON, TOF, RTO
Counter Instructions CTU, CTD
Reset RES

Using Timers and Counters

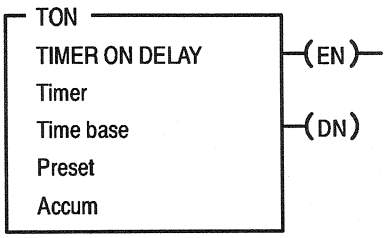
Timers and counters let you control operations based on time or number of events. Table 2.A lists the available timer and counter instructions.

Table 2.A
Available Timer and Counter Instructions

If You Want to:	Use This Instruction:	Found on Page:
Delay turning on an output	TON	2-3
Delay turning off an output	TOF	2-6
Time an event retentively	RTO	2-8
Count up	CTU	2-12
Count down	CTD	2-14
Reset a counter, timer, or counter-driven instruction	RES	2-18

Using Timers

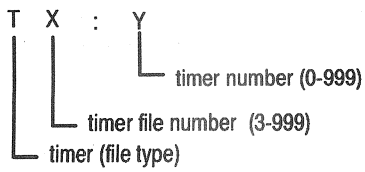
Before you program timer instructions, you need to understand the parameters that you enter for timer instructions and how timer accuracy works.



Entering Parameters

To program a timer instruction, you need to provide the processor with the following information:

- **Timer** is the timer control address in the timer (T) area of data storage. Use the following address format:



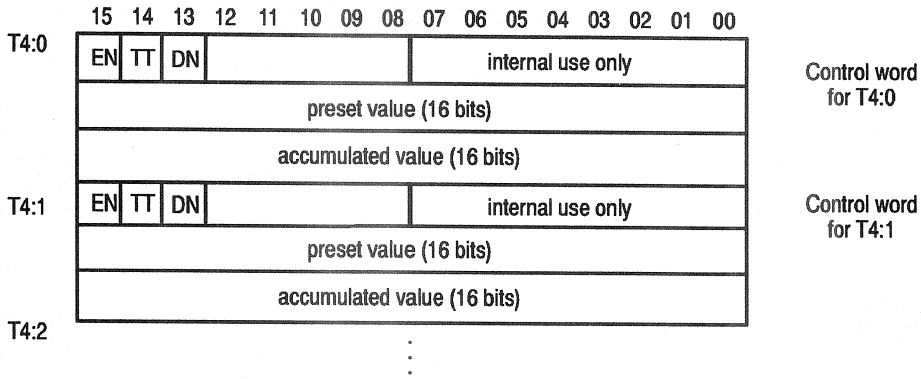
Important: Note that you can use any timer file number from 3 to 999; however, the default timer file number is 4. If you want to specify a timer file number as any file between 3 and 8 (other than the default 4), you must first delete the entire default file for that number, and then create the timer file. For example, if you want a timer file number as file 3, you must first delete the entire default binary file and then create the timer file as file 3.

Whenever you want your ladder program to access a timer status bit, preset, or accumulated value stored at the timer control address, use the following address format:

Status Bit	Preset	Accumulated Value
TX:Y.*sb*	TX:Y.PRE	TX:Y.ACC

The *sb* specifies a status bit mnemonic, such as .DN

Important: When you program a timer instruction, the processor stores status bits and the preset and accumulated values in a 48-bit storage structure (three 16-bit words) in the timer area (T) of data memory.



- **Time Base** determines how the timer operates. Table 2.B lists the possible time bases.

Table 2.B
Available Time Base Values

Enter This Time Base:	The Accumulated Value Range Is:
1 second	to 32,767 time-base intervals (to 9.1hours)
0.01 seconds (10ms)	to 32,767 time-base intervals (to 5.5 minutes)

- **Preset** specifies the value which the timer must reach before the processor sets the done bit (.DN) so your ladder program can take some action. When you program a timer instruction, you must enter a preset value from 0–32,767. The processor stores the preset value as a 16-bit integer value.

  Important: The Preset value operates differently if you are using a TOF instruction. See page 2-6 for more information.

- **Accumulated Value** is the number of time increments the instruction has counted. When enabled, the timer updates this value continually. Typically, enter zero when programming the instruction. If you enter a value, the instruction starts counting time base intervals from that value. If the timer is reset, the accumulated value is zero. The range for the accumulated value is 0–32,767. The processor stores the accumulated value as a 16-bit integer.

  Important: The Accumulated value operates differently if you are using a TOF instruction. See page 2-6 for more information.

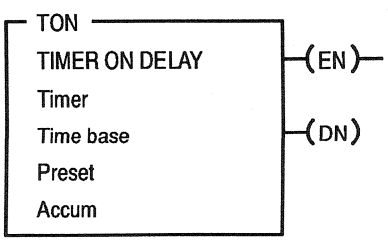
Timer Accuracy

Timer accuracy refers only to the length of time between the moment the processor enables a timer instructions and the moment the processor completes the timed interval. Timing accuracy depends on the processor clock tolerance and the time base. The clock tolerance is ± 0.02%. This means that a timer could time out early or late by 0.01 seconds (10ms) for a 0.01 second time base or 1 second for a 1 second time base.

The 0.01-second timer maintains accuracy with a program scan of up to 2.5 seconds; the 1-second timer maintains accuracy with a program scan of up to 1.5 seconds. If your programs can exceed 1.5 or 2.5 seconds, repeat the timer instruction rung so that the rung is scanned within these limits.

The displayed accumulated value of a timer shows actual time but is dependent on CRT update time. The accumulated value might appear to be less than the preset when the done bit is set.

Timer On Delay (TON)



Description:
Use the TON instruction to turn an output on or off after the timer has been on for a preset time interval. The TON instruction starts accumulating time when the rung goes true, and continues until one of the following happens:

- the accumulated value equals its preset value
- the rung goes false
- a reset instruction resets the timer
- the associated SFC step goes inactive

The processor resets the accumulated value when the rung conditions go false, regardless of whether the timer timed out or not.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address status bits by mnemonic.

This Bit:	Is Set When:	And Remains Set Until:
Timer Enable .EN (bit 15) When set, the .EN bit indicates that the timer is enabled.	the rung goes true	• the rung goes false • a reset instruction resets the timer • the associated SFC step goes inactive
Timer Timing Bit .TT (bit 14) When set, the .TT bit indicates that a timing operation is in progress.	the rung goes true	• the rung goes false • the .DN bit is set (.ACC = .PRE) • a reset instruction resets the timer • the associated SFC step goes inactive
Timer Done Bit .DN (bit 13) When set, the .DN bit indicates that a timing operation is complete.	the accumulated value is equal to the preset value	• the rung goes false • a reset instruction resets the timer • the associated SFC step goes inactive

If you set the done bit .DN using an OTE instruction, for example, you can pause the timer. The .EN and .TT bits remain set, but the accumulated value does not increment. Timing resumes when you clear the .DN bit. If the rung goes false while the timer is paused, the timer resets as normal.

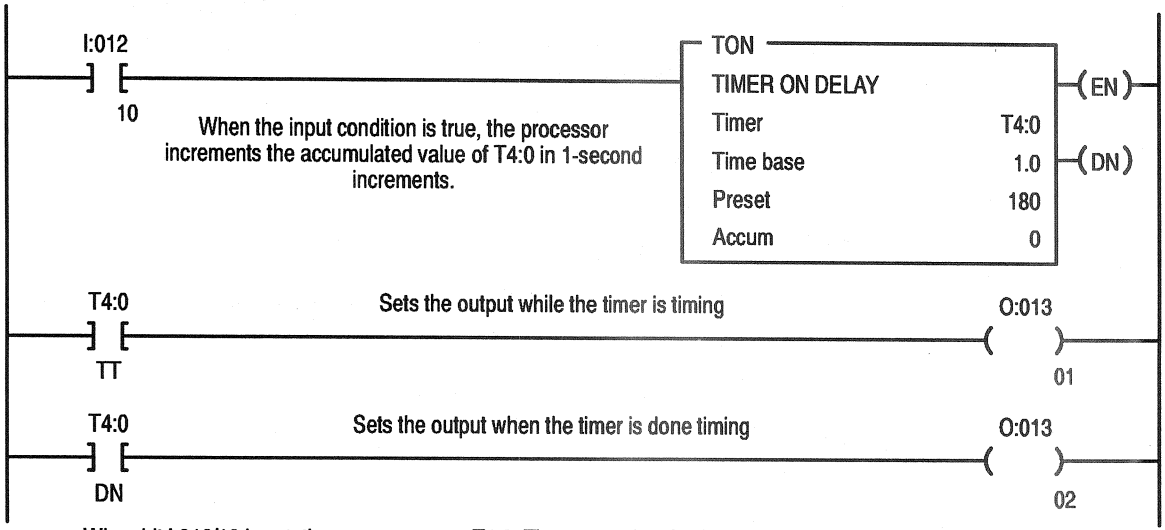
If you change to Program mode, or the processor loses power before the instruction reaches the preset value, the following occurs:

- timer enable (.EN) bit remains set
- timer timing (.TT) bit remains set
- accumulated (.ACC) value remains the same

Then when you switch back to Run mode or Test mode, the following happens:

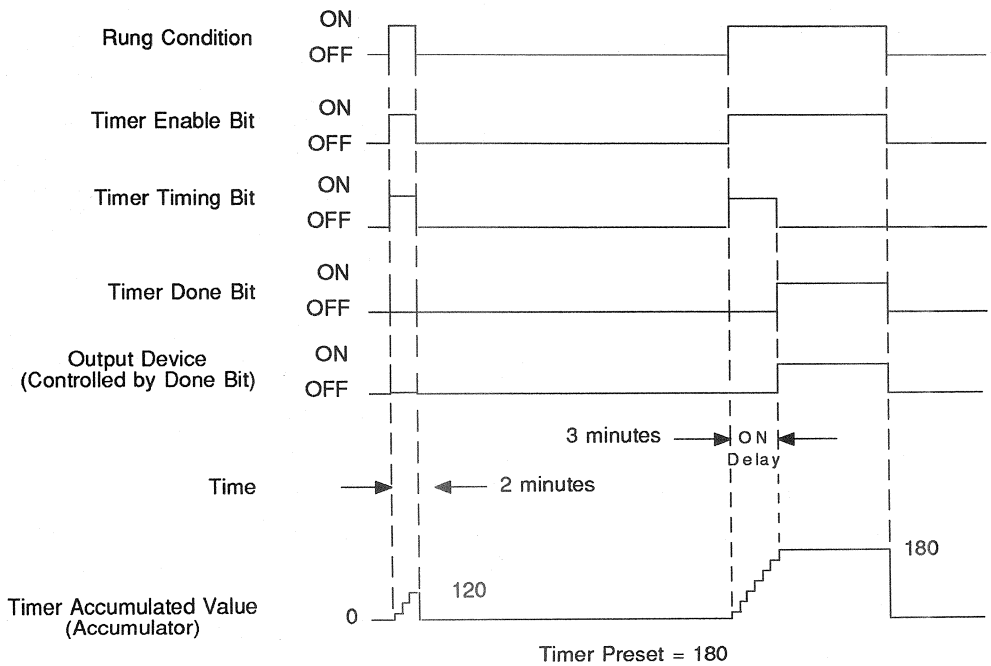
Condition:	Result:
If the rung is true:	.EN bit remains set .TT bit remains set .ACC value is reset and starts counting up
If the rung is false:	.EN bit is reset .TT bit is reset .DN bit is reset .ACC value is reset

Figure 2.1
Example TON Ladder Diagram



When bit I:012/10 is set, the processor start T4:0. The accumulated value increments in 1-second intervals. T4:0.TT is set and output bit O:013/01 is set (the associated output device is energized) while the timer is timing. When the timer is finished (.ACC = .PRE) T4:0.TT is reset (so O:013/01 and the associated output device is de-energized) and T4:0.DN is set (so O:013/02 is set and the associated output device is energized). When the accumulated value reaches 180 or when the rung conditions go false, the timer stops.

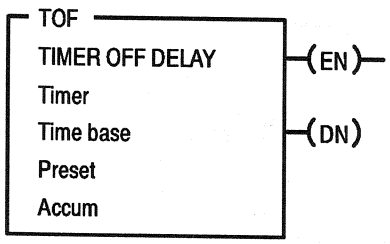
Figure 2.2
Example TON Timing Diagram



TON Timing Diagram

16649

Timer Off Delay (TOF)



Description:
Use the TOF instruction to turn an output on or off after its rung has been off for a preset time interval. The TOF instruction starts accumulating time when the rung goes false and continues timing until one of the following conditions occur:

- the accumulated value equals its preset value
- the rung goes true
- a reset instruction resets the timer
- the associated SFC step goes inactive

The processor resets the accumulated value when the rung conditions go true, regardless of whether the timer timed out or not.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address the status bits by mnemonic.

This Bit:	Is Set When:	And Remains Set Until:
Timer Enable .EN (bit 15)	the rung goes true	• the rung goes false • a reset instruction resets the timer • the associated SFC step goes inactive
Timer Timing Bit .TT (bit 14)	the rung goes false and the accumulated value is less than the preset	• the rung goes true • the .DN bit is set (.ACC = .PRE) • a reset instruction resets the timer • the associated SFC step goes inactive
Timer Done Bit .DN (bit 13)	the accumulated value is equal to the preset value	• the rung goes true • a reset instruction resets the timer • the associated SFC step goes inactive

If the processor loses power:

1. Change to Program mode, or the processor fault interrupts the TOF instruction before it reaches the preset value, the following occurs:
   - timer enable (.EN) bit remains reset
   - timer timing (.TT) bit remains set
   - timer done (.DN) bit remain set
   - accumulated (.ACC) value remains the same

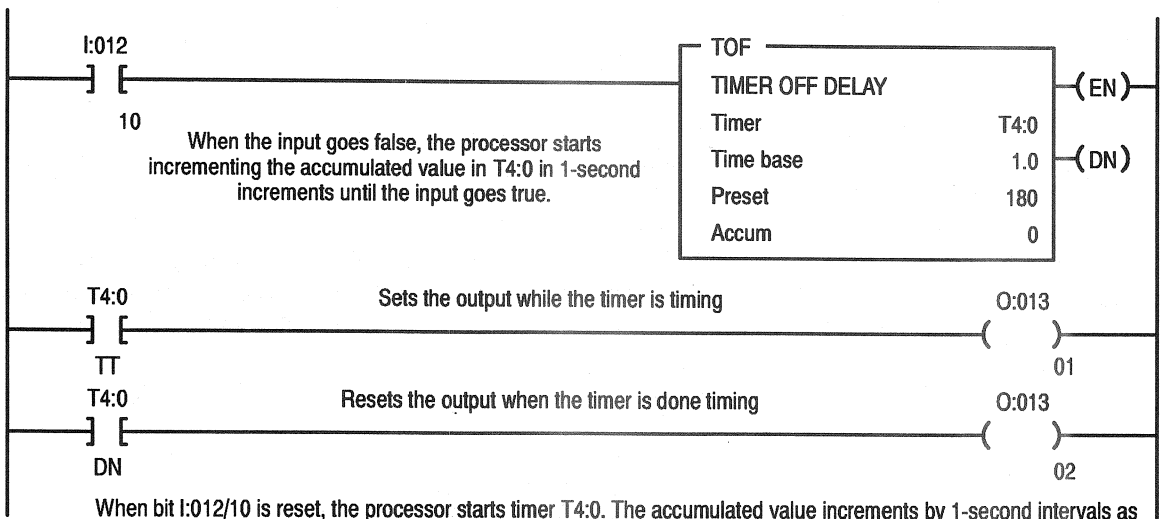
2. Then if you switch to Run mode or Test mode, the following happens:

Condition:	Result:
If the rung is true:	.EN bit is set .TT bit is reset .DN bit remains set .ACC value is cleared
If the rung is false	.EN bit is reset .TT bit is reset .DN bit is reset .ACC value equals PRE value



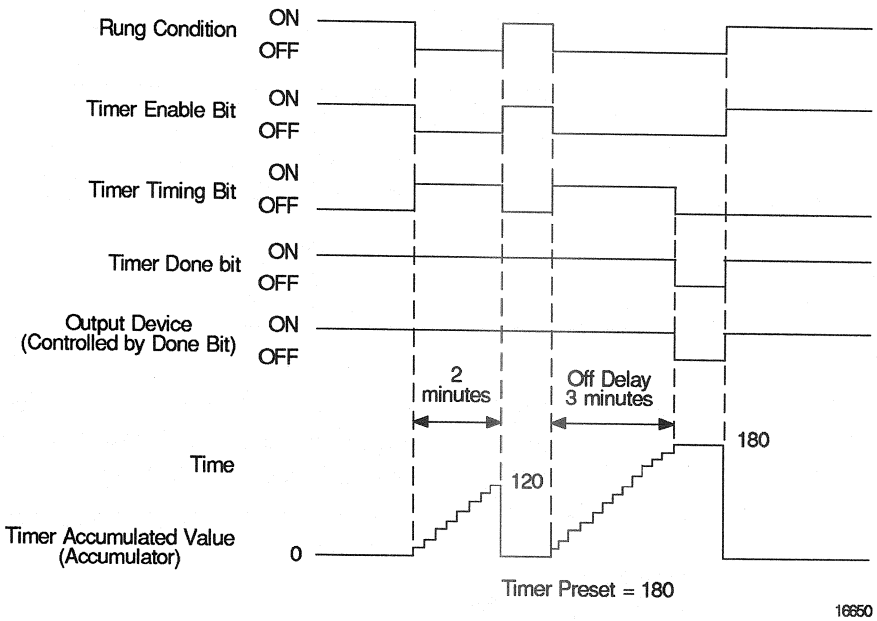
WARNING: Because the RES instruction resets the accumulated value, done bit and timing bits of a timing instruction, do not use the RES instruction to reset a TOF timer.

Figure 2.3
Example TOF Ladder Diagram

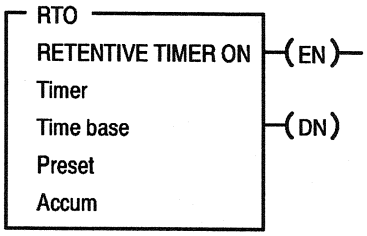


When bit I:012/10 is reset, the processor starts timer T4:0. The accumulated value increments by 1-second intervals as long as the rung remains false. T4:0.TT is set and output bit O:013/01 is set (the associated output device is energized) while the timer is timing. When the timer is finished (.ACC = .PRE), T4:0.TT is reset (so O:013/01 is reset and the associated output device is de-energized) and T4:0.DN is reset (so O:013/02 is reset and the associated output device is de-energized). When the accumulated value reaches 180 or when the rung conditions go true, the timer stops.

Figure 2.4
Example TOF Timing Diagram



Retentive Timer On (RTO)



Description:
Use the RTO instruction to turn an output on or off after its timer has been on for a preset time interval. The RTO instruction lets the timer stop and start without resetting the accumulated value.

The RTO instruction begins timing when its rung goes true. As long as the rung remains true, the timer updates the accumulated value each program scan, until it reaches the preset value. The RTO instruction retains its accumulated value even if one of the following occurs:

- the rung goes false
- you change to Program mode
- the processor faults or loses power
- the associated SFC step goes inactive

When the processor resumes operation or the rung goes true, timing continues from the retained accumulated value. By retaining its accumulated value, retentive timers measure the cumulative period during which its rung is true.

Important: To reset the retentive timer's accumulated value and status bits after the RTO rung goes false, you must program a reset instruction RES with the same address in another rung.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address the status bits by mnemonic.

This Bit:	Is Set When:	And Remains Set Until:
Timer Enable .EN (bit 15) When set, the .EN bit indicates that a timing operation is in progress.	the rung goes true	• the rung goes false • a reset instruction resets the timer
Timer Timing Bit .TT (bit 14) When set, the .TT bit indicates that a timing operation is in progress.	the rung goes true	• the rung goes false • the .DN bit is set • the accumulated value is equal to the preset value (.ACC=.PRE) • a reset instruction resets the timer
Timer Done Bit .DN (bit 13) When set, the .DN bit indicates that a timing operation is complete.	the accumulated value is equal to the preset value	The .DN bit is reset with the RES instruction.

If the processor loses power:

1. Change to Program mode, or processor fault interrupts the RTO instruction, the following occurs:
   - timer enable (.EN) bit remains set
   - timer timing (.TT) bit remains set
   - accumulated (.ACC) value remains the same
2. Then when you switch back to Run mode or Test mode, the following happens:

Condition:	Result:
If the rung is true:	.EN bit remains set .TT bit remains set .ACC value continues timing
If the rung is false:	.EN bit is reset .TT bit is reset .DN bit is reset .ACC value remains the same

Figure 2.5
Example RTO Ladder Diagram

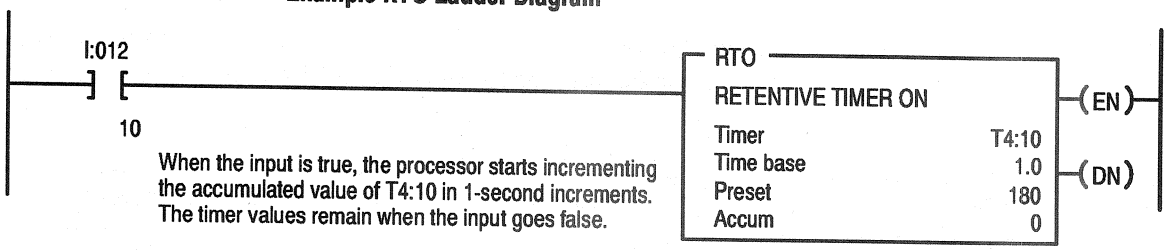
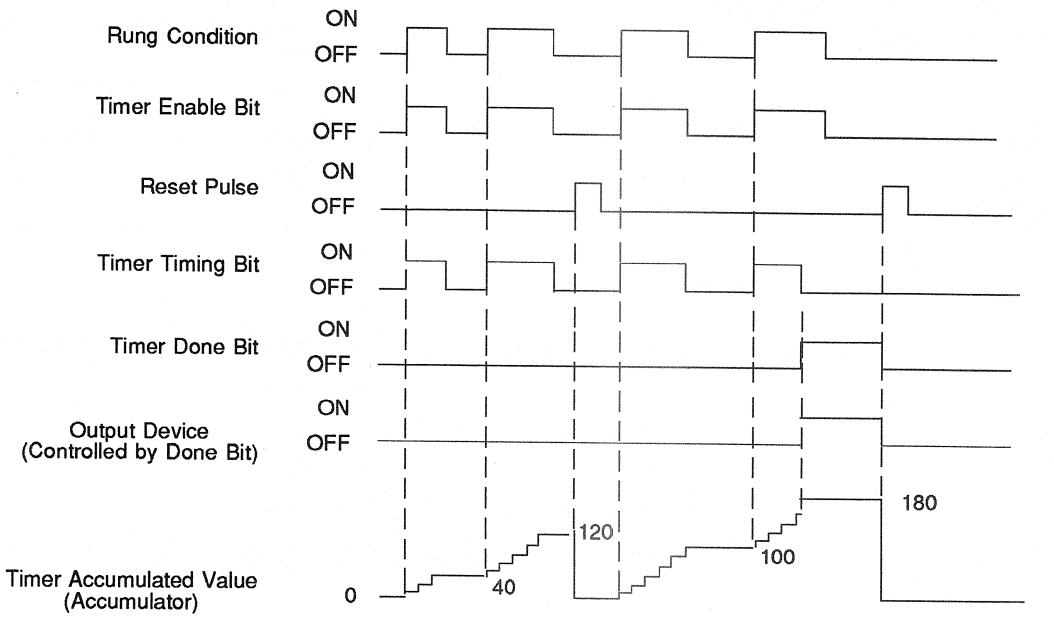
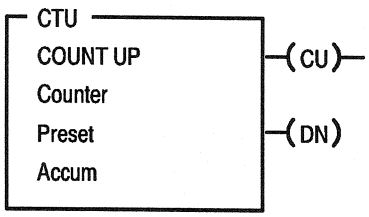


Figure 2.6
Retentive Timer Timing Diagram



Using Counters

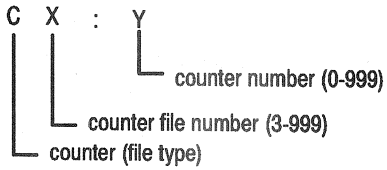


Before using counter instructions, you need to understand the parameters that you enter.

Entering Parameters

To program a timer instruction, you need to provide the processor with the following information:

- **Counter** is the counter control address in the counter (C) area of data storage. Use the following address format:



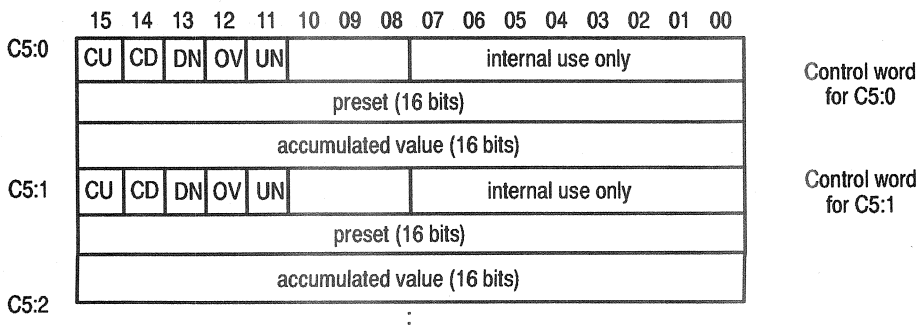
Important: Note that you can use any counter file number from 3 to 999; however, the default counter file number is 5. If you want to specify a counter file number as any file between 3 and 8 (other than the default 5), you must first delete the entire default file for that number, and then create the counter file. For example, if you want a counter file number as file 3, you must first delete the entire default binary file and then create the counter file as file 3.

When you want your ladder program to access a counter status bit, preset value, or accumulated value, use the following address format:

Status Bit	Preset	Accumulated Value
CX:Y.*sb*	CX:Y.PRE	CX:Y.ACC

The *sb* is a status bit mnemonic, such as .DN

Important: When you program a counter instruction, the processor stores status bits and the preset and accumulated values in a storage structure (48 bits – three 16-bit words) in the counter area (C) of data memory.

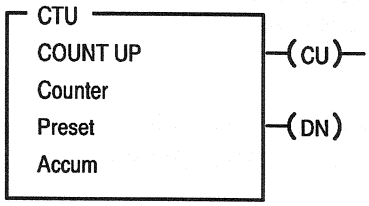


- **Preset** specifies the value which the counter must reach before it sets the done bit .DN so your ladder program can take some action. When you program a counter instruction, you must enter a preset value from -32,768 up to +32,767. The preset value is stored as a 16-bit integer value. Negative values are stored in two's complement form.

- **Accumulated Value** is the current count based on the number of times the rung goes from false to true. The accumulated value is stored as a 16-bit integer value. Negative values are stored in two's complement form. The range of the accumulated value is –32,768 to +32,767.

  Typically, you enter a zero value when programming counter instructions. If you enter a non-zero value, the instruction starts counting from that value. If the counter is reset the accumulated value is set to zero.

Count Up (CTU)



Description:
The CTU instruction counts upward over a range of -32,768 to +32,767. Each time the rung goes from false to true, the CTU instruction increments the accumulated value by one count. When the accumulated value equals or exceeds the preset value, the CTU instruction sets a done bit .DN, which your ladder program can use to initiate some action, such as controlling a storage bit or an output device.

The accumulated value of a counter is retentive. The count is retained until reset by a reset instruction (RES) that has the same address as the counter.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs the CTU instruction. You address the status bits by mnemonic.

- **Count Up Enable Bit .CU** (bit 15) is set when the rung goes true to indicate that the counter is enabled as an up-counter. The .CU bit is reset when the rung goes false or when it is reset by a RES instruction.
- **Count Up Done Bit .DN** (bit 13) is set by the processor to show when the accumulated value reaches the preset value, and remains set when the accumulated value exceeds the preset. If you combine a CTU and CTD instruction, counting back down past the preset value resets the done bit. Otherwise, the .DN bit is reset by the RES instruction.
- **Count Up Overflow Bit .OV** (bit 12) is set by the processor to show that the count exceeded the upper limit of +32,767, and has wrapped around to -32,768. The CTU instruction counts up from there.

The CTU instruction can count beyond the preset value. When it reaches 32,767 + 1, it sets the .OV bit. You can reset this bit with a RES instruction that has the same address as the counter, or by counting back down to 32,767 with a CTD instruction that has the same address.

WARNING: Place critical counters outside an MCR zone or jumped sections of ladder program to guard against invalid results that could lead to damaged equipment or personnel injury.

Figure 2.7
Example CTU Ladder Diagram

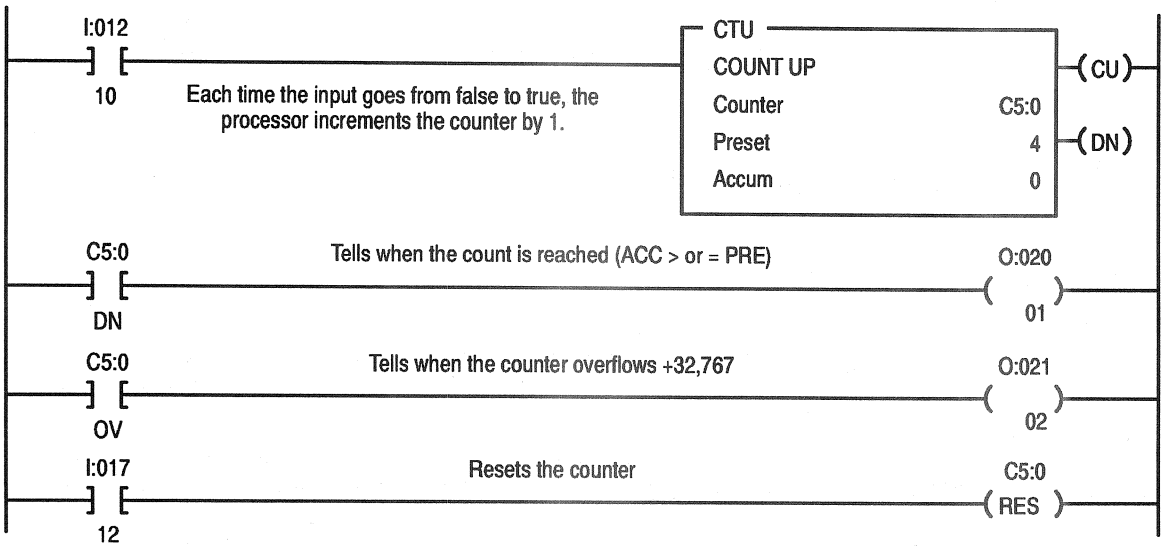
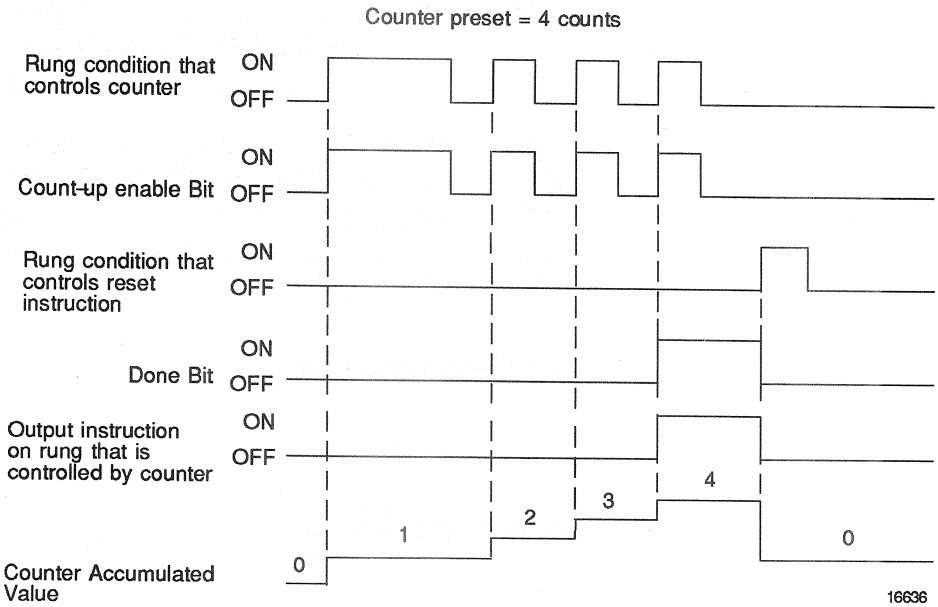
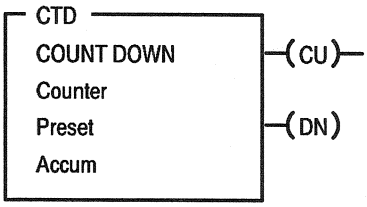


Figure 2.8
Example CTU Timing Diagram



Count Down (CTD)



Description:
The CTD instruction counts downward over a range of +32,767 to -32,768. Each time the rung goes from false to true, the CTD instruction decrements the accumulated value by one count. The done bit .DN is set as long as the accumulated value is greater than or equal to the preset value. When the accumulated value is less than the preset value, the done bit .DN is reset, which your ladder program can use to initiate some action, such as controlling a storage bit or an output device.

The accumulated value of a counter is retentive. The count is retained until reset by a reset instruction (RES) that has the same address as the CTD instruction.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address the status bits by mnemonic.

- **Count Down Enable Bit .CD** (bit 14) is set when the rung goes true to indicate that the counter is enabled as a down-counter. The .CD bit is reset when the rung goes false or is reset by a RES instruction.

- **Count Down Done Bit .DN** (bit 13) is set by the processor when the accumulated value is greater than or equal to the preset value. The .DN bit is reset when the accumulated value counts below the preset value. If you combine a CTU and CTD instruction, counting back up past the preset value sets the done bit. The done bit is reset by the RES instruction.
- **Count Down Underflow Bit .UN** (bit 11) is set by the processor to show that the down counter has exceeded the lower limit of -32,768 and has wrapped around to +32,767. The CTD instruction counts down from there.

  You can reset this bit with a RES instruction that has the same address as the CTD instruction, or you can count back up to -32,768 with a CTU instruction that has the same address as the CTD instruction.



WARNING: Place critical counters outside an MCR zone or jumped sections of ladder program to guard against invalid results that could lead to damaged equipment or personnel injury.

Figure 2.9
Example CTD Ladder Diagram

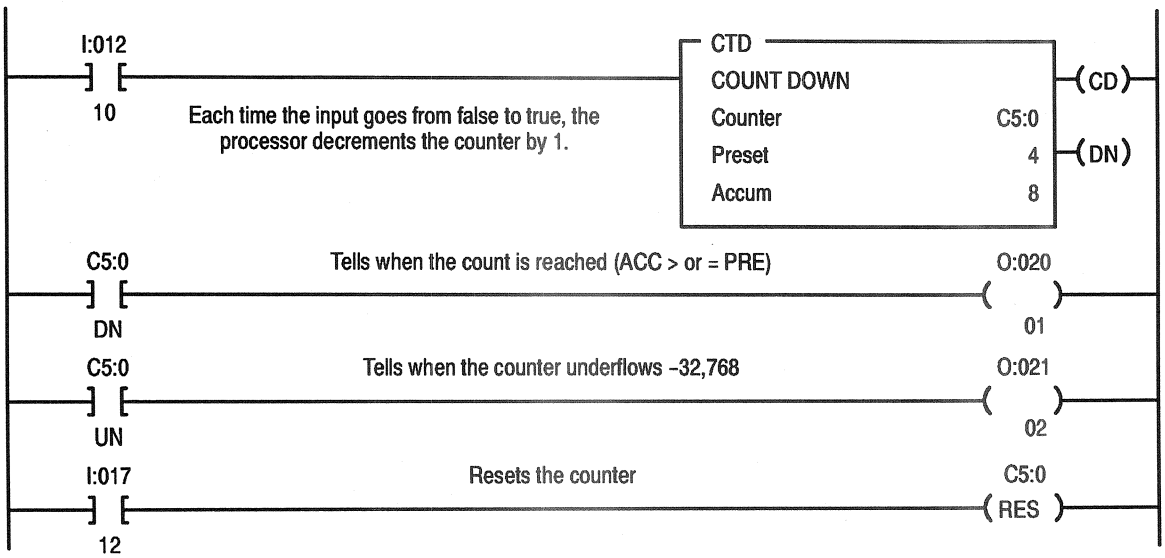


Figure 2.10
Example CTD Timing Diagram

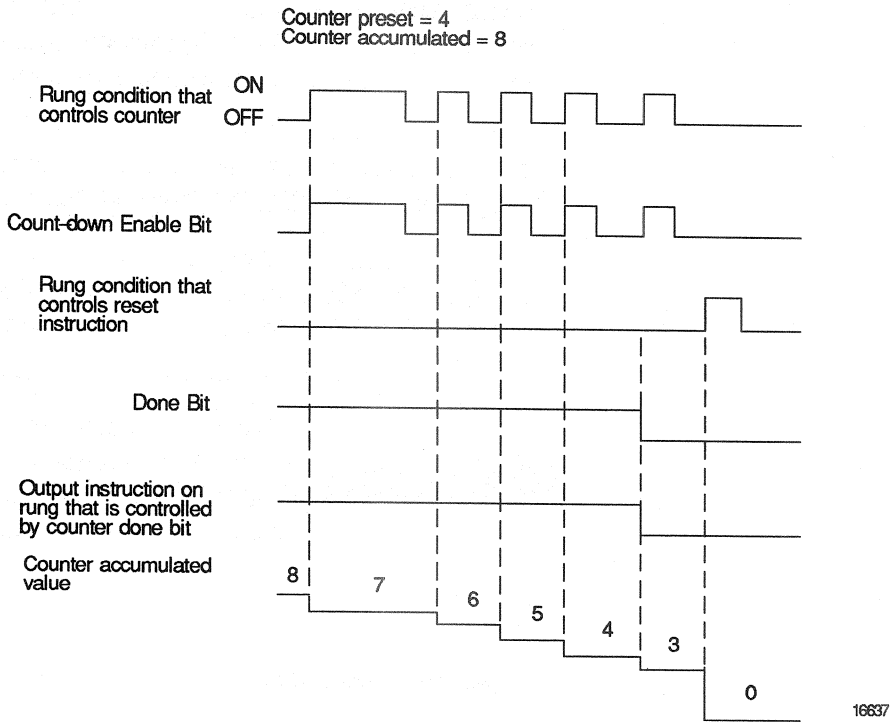


Figure 2.11
Example CTU and CTD Logic Diagram

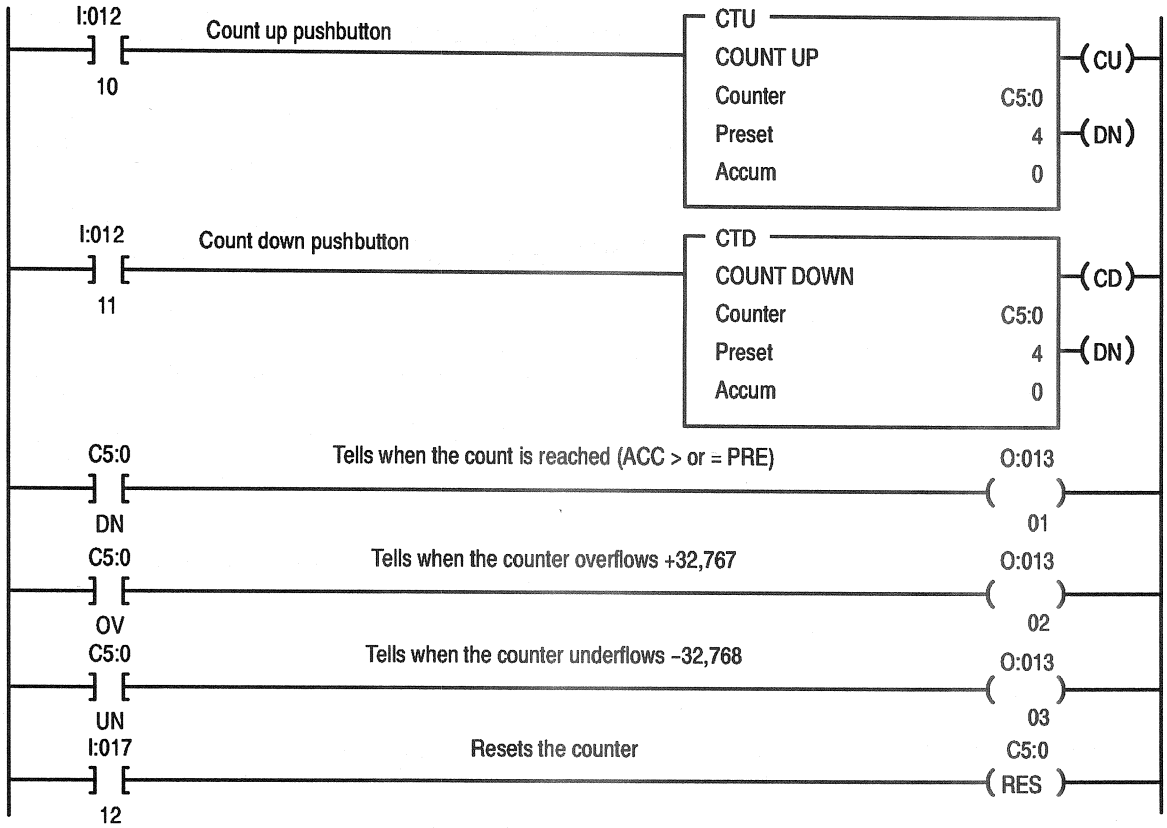
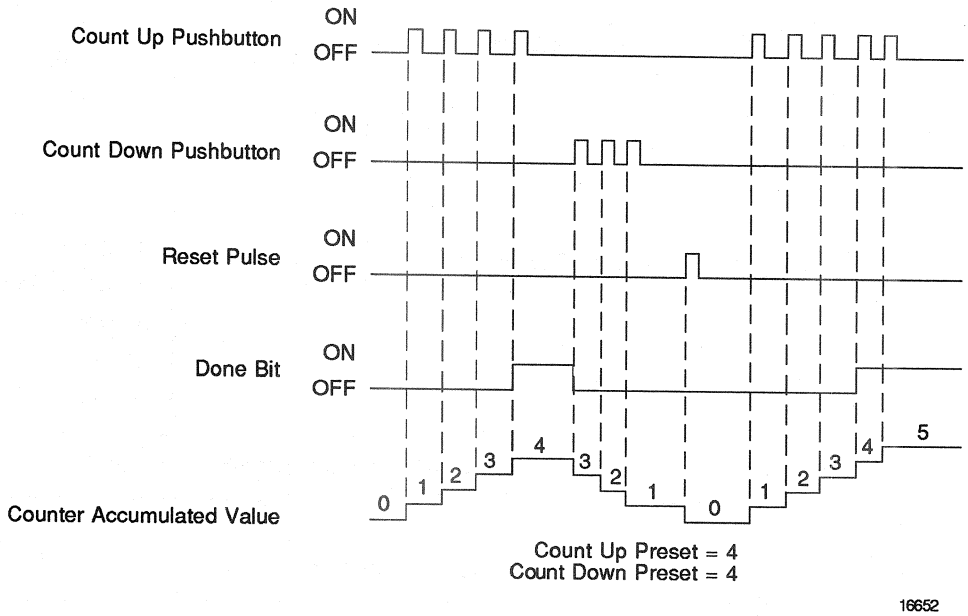


Figure 2.12
Example CTU and CTD Timing Diagram



Timer and Counter Reset (RES)

—(RES)—

Description:
The RES instruction is an output instruction that you enter to reset a timer or counter. The RES instruction executes when its rung is true.

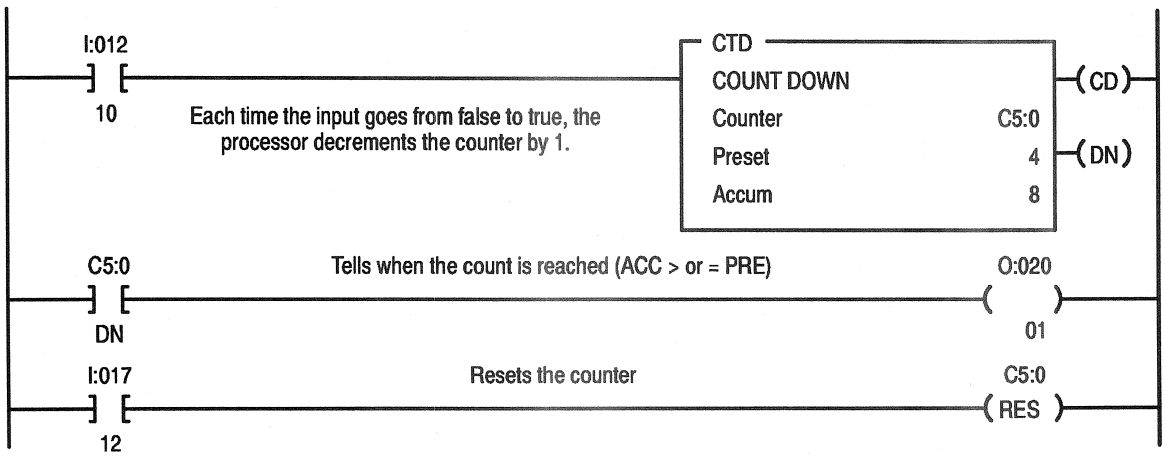
When Using a RES Instruction for a:	The Processor Resets the:
Timer (Do not use a RES instruction for a TOF.)	.ACC value .EN bit .TT bit .DN bit
Counter	.ACC value .EN bit .OV or .UN bit .DN bit

If the counter rung is enabled, the CU or CD bit will be reset as long as the RES instruction is enabled.

Important: You can use a negative preset value in a CTU or CTD instruction if you intend to use the RES instruction. However, note that the RES instruction sets the accumulated value to zero, which may set the .DN bit and prevent the CTU or CTD instruction from operating the next time it is enabled.

WARNING: Because the RES instruction resets the accumulated value, done bit and timing bit of a timing instruction, do not use the RES instruction to reset a TOF instruction; unpredictable machine operation or injury to personnel may occur

Figure 2.13
Example RES Ladder Diagram



Chapter 3

Compare Instructions CMP, EQU, GEQ, GRT, LEQ, LES, LIM, MEQ, NEQ

Using Compare Instructions

The comparison instructions let you compare values using an expression or using a specific comparison instruction. Table 3.A lists the available compare instructions.

Table 3.A
Available Compare Instructions

If You Want to:	Use this Instruction:	Found on Page:
Compare values based on an expression	CMP	3-2
Test whether two values are equal	EQU	3-4
Test whether one value is greater than or equal to a second value	GEQ	3-5
Test whether one value is greater than a second value	GRT	3-5
Test whether one value is less than or equal to a second value	LEQ	3-5
Test whether one value is less than a second value	LES	3-6
Test whether one value is within the limit range of two other values	LIM	3-6
Pass two values through a mask and test whether they are equal	MEQ	3-7
Test whether one value is not equal to a second value	NEQ	3-9

Important: You can compare values of different data types, such as floating point and integer. You should use BCD and ASCII values for display purposes. If you enter BCD or ASCII values, the processor treats those values as integers. For example, if the value at N7:2 is 10 (decimal) and the value at D9:3 is 10 (BCD), the comparison of N7:2 = D9:3 evaluates as false. The 10 in BCD translates to 0000 0000 0001 0000; the 10 in decimal translates to 0000 0000 0000 1010.

The parameters you enter are program constants or logical addresses of the values you want to compare.

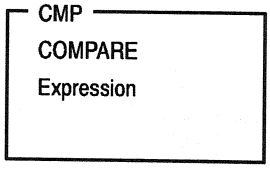
Using Arithmetic Status Flags

The arithmetic status flags are in word 0 bits 0–3 in the processor status file (S). You can monitor these bits if you are performing an arithmetic function within the CMP instruction. Table 3.B lists the status bits:

Table 3.B
Arithmetic Status Bits

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

Compare (CMP)



The CMP instruction compares values and performs logical comparisons.

Description

The CMP instruction is an input instruction that performs a comparison on arithmetic operations you specify in the expression. When the processor finds the expression is true, the rung goes true. Otherwise, the rung is false. With PLC-5/30, -5/40 and -5/60 processors, you can enter multiple operands (complex expression).

The execution time of a CMP instruction is longer than the execution time of one of the other comparison instructions (e.g., GRT, LEQ, etc.). A CMP instruction also uses more words in your program file than the corresponding comparison instruction.

Entering the CMP Expression

The expression defines the operations that you want to perform. You define the expression with operators and addresses or program constants. With PLC-5/30, -5/40 and -5/60 processors, you can enter complex expressions (up to a total of 80 characters). Table 3.C lists valid operations for an expression; the following list provides guidelines for writing expressions.

- Operators (symbols) define the operations
- Addresses can be direct, indirect, or indexed address(es) (must be word level)
- With PLC-5/30, -5/40 and -5/60 processors, program constants can be integer or floating-point numbers (if you enter octal values, use a leading &O; if you enter hexadecimal values, use a leading &H; if you enter binary values, use a leading &B)

Table 3.C
Valid Operations for Use in a CMP Expression

Type	Operator	Description	Example Operation
Comparison	=	equal to	if A = B, then ...
	<>	not equal to	if A <> B, then ...
	<	less than	if A < B, then ...
	<=	less than or equal to	if A <= B, then ...
	>	greater than	if A > B, then ...
	>=	greater than or equal to	if A >= B, then ...
Arithmetic	+	add	2 + 3 PLC-5/30, -5/40, -5/60: 2 + 3 + 7
	-	subtract	12 - 5
	*	multiply	5 * 2 PLC-5/30, -5/40, -5/60: 6 * (5 * 2)
	| (vertical bar)	divide	24 | 6
	-	negate	- N7:0
	SQR	square root	SQR N7:0
	**	exponential (x to the power of y)	10**3 (PLC-5/30, -5/40, -5/60 only)
Conversion	FRD	convert from BCD to binary	FRD N7:0
	TOD	convert from binary to BCD	TOD N7:0

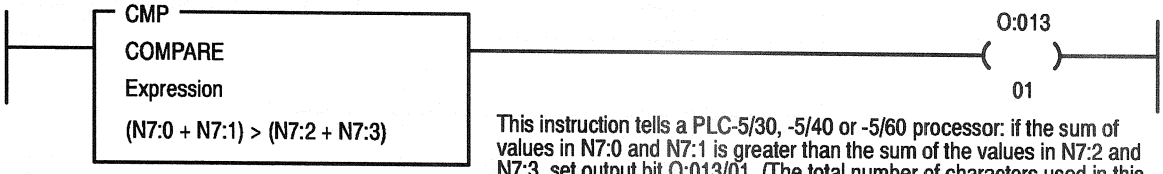
Determining the Length of an Expression

With PLC-5/30, -5/40, and -5/60 processors, you can enter complex instructions (up to a total of 80 characters, including spaces and parentheses). Depending on the operator, the processor inserts characters before/after the operator in your expression to format the expression for easier interpretation. Use Table 3.D to determine the number of characters each operator uses in an expression.

Table 3.D
Character Lengths for Operators

This Operation:	Using This Operator:	Uses This Number of Characters:
math binary	+, -, *, |	3
	OR, **	4
	AND, XOR	5
math unary	- (negate)	2
	LN	3
	FRD, TOD, DEG, RAD, SQR, NOT, LOG, SIN, COS, TAN, ASN, ACS, ATN	4
comparative	=, <, >	3
	<>, <=, >=	4

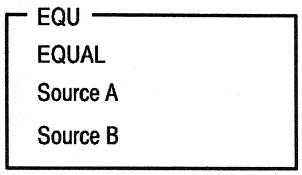
Example:



This instruction tells a PLC-5/30, -5/40 or -5/60 processor: if the sum of values in N7:0 and N7:1 is greater than the sum of the values in N7:2 and N7:3, set output bit O:013/01. (The total number of characters used in this expression is 29, according to Table 3.D above.)

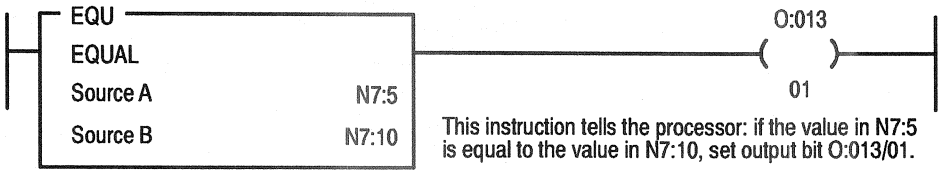
For more information on entering complex expressions, refer to the "Compute Instructions" chapter in this module.

Equal to (EQU)



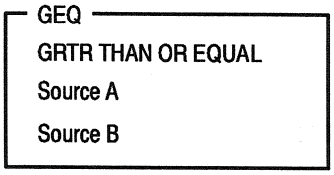
Description:
Use the EQU instruction to test whether two values are equal. Source A and Source B can either be values or addresses that contain values.

Example:



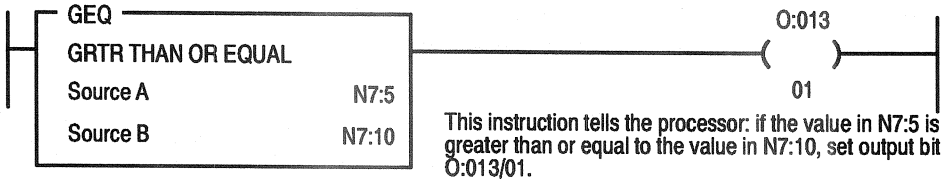
This instruction tells the processor: if the value in N7:5 is equal to the value in N7:10, set output bit O:013/01.

Greater than or Equal to (GEQ)



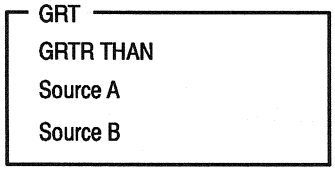
Description:
Use the GEQ instruction to test whether one value (Source A) is greater than or equal to another value (Source B). Source A and Source B can either be values or addresses that contain values.

Example:



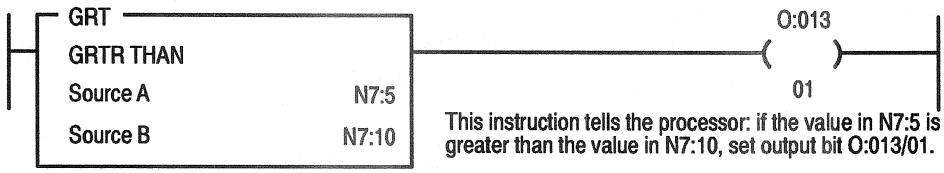
This instruction tells the processor: if the value in N7:5 is greater than or equal to the value in N7:10, set output bit O:013/01.

Greater than (GRT)



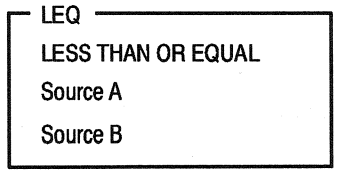
Description:
Use the GRT instruction to test whether one value (Source A) is greater than another value (Source B). Source A and Source B can either be values or addresses that contain values.

Example:



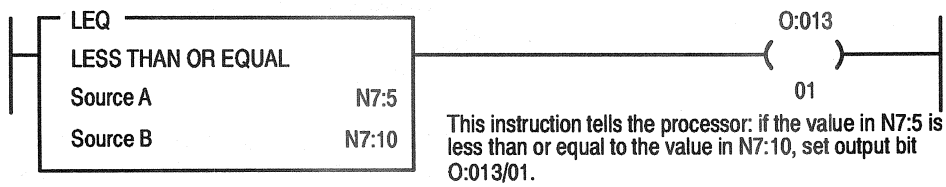
This instruction tells the processor: if the value in N7:5 is greater than the value in N7:10, set output bit O:013/01.

Less than or Equal to (LEQ)



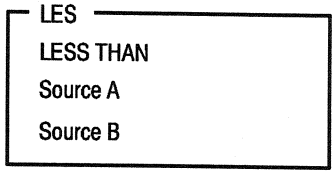
Description:
Use the LEQ instruction to test whether one value (Source A) is less than or equal to another value (Source B). Source A and Source B can either be values or addresses that contain values.

Example:



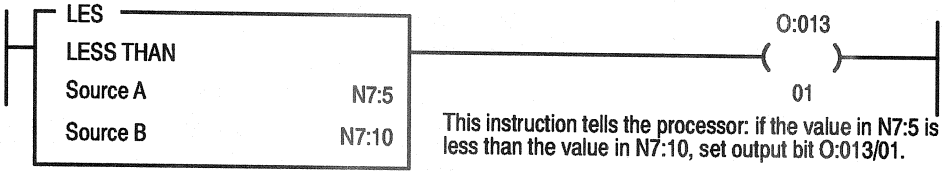
This instruction tells the processor: if the value in N7:5 is less than or equal to the value in N7:10, set output bit O:013/01.

Less than (LES)

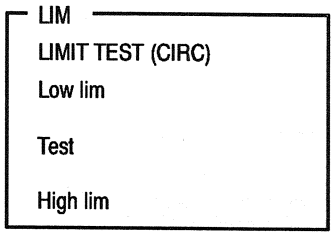


Description:
Use the LES instruction to test whether one value (Source A) is less than another value (Source B). Source A and Source B can either be values or addresses that contain values.

Example:



Limit Test (LIM)



Description:
The LIM instruction is an input instruction that tests for values inside of or outside of a specified range. The instruction is false until it detects that the test value is within certain limits. Then the instruction goes true. When the instruction detects that the test value goes outside certain limits, it goes false.

You can use the LIM instruction to test if an analog input value is within specified limits.

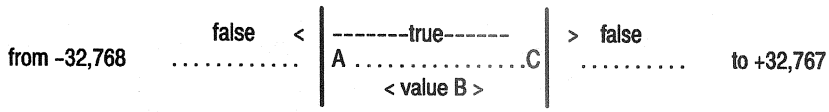
Entering Parameters

To program the LIM instruction, you must provide the processor with the following:

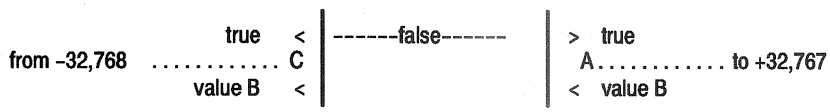
- **Low Limit** is a constant or an address from which the instruction reads the lower range of the specified limit range. The address contains an integer or floating-point value.
- **Test Value** is the address that contains the integer or floating-point value you examine to see whether the value is inside or outside the specified limit range.
- **High Limit** is a constant or an address from which the instruction reads the upper range of the specified limit range. The address contains an integer or floating-point value.

LIM Examples:

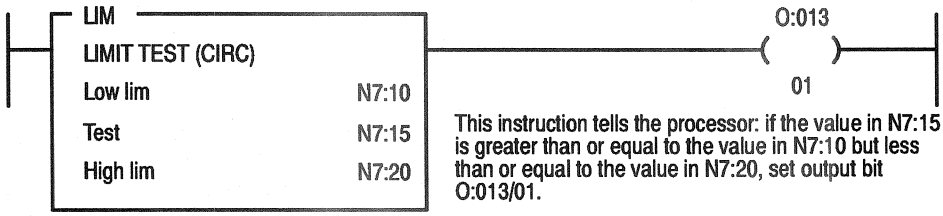
If value A ≤ value C: When the processor detects that the value of B (Test) is equal to or between limits (Low Limit = A, High Limit = C), the instruction is true; if value B is outside the limits, the instruction is false.



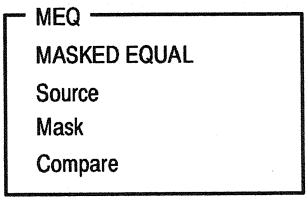
If value A ≥ value C: When the processor detects that the value of B(Test) is equal to or outside the limits (Low Limit = A, High Limit = C), the instruction is true; if value B is between, but not equal to either limit, the instruction is false.



Example (when the Low Limit is less than the High Limit):



Mask Compare Equal to (MEQ)



Description:
The MEQ instruction is an input instruction that compares a value from a source address with data at a compare address, and allows portions of the data to be masked.

If the data at the source address matches the data at the compare address bit-by-bit (less masked bits), the instruction is true. The instruction goes false as soon as it detects a mismatch.

You can use the MEQ instruction to extract (for comparison) bit data such as status or control bits from an element that contains bit and word data.

Entering Parameters

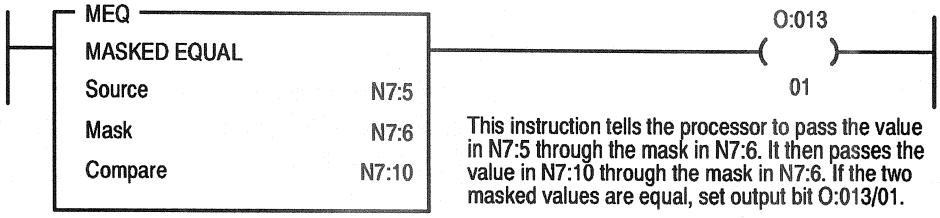
To program this instruction, you must provide the processor with the following:

- **Source** is a program constant or data address from which the instruction reads an image of the value. The source remains unchanged.
- **Mask** specifies which bits to pass or block. A mask passes data when the mask bits are set (1); a mask blocks data when the mask bits are reset (0). The mask must be the same element size (16-bits) as the source and compare address. You must set (1) mask bits to compare data. Bits in the compare address that correspond to zeros in the mask are not compared.

  If you want the ladder program to change the mask value, store the mask at a data address. Otherwise, enter a hexadecimal value for a constant mask value. If you enter a hexadecimal value that starts with a letter (such a F800), enter the value with a leading zero. For example, type `0F800`
- **Compare** specifies whether you want the ladder program to vary the compare value, or a program constant for a fixed reference. Use 16-bit elements, the same as the source.

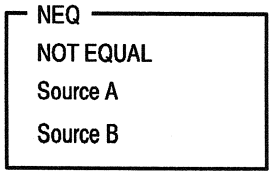
Example:

```
Source      01010101 01011111
Mask        11111111 11110000
Compare     01010101 0101xxxx
Result      The instruction is true because
            reference bits xxxx are not compared.
```



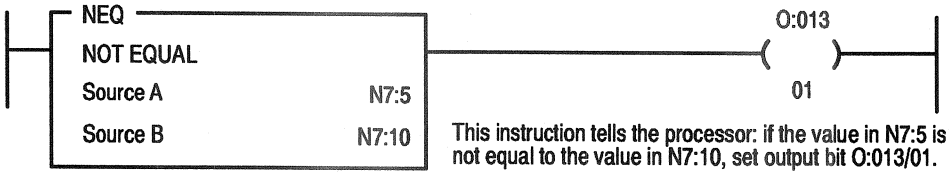
This instruction tells the processor to pass the value in N7:5 through the mask in N7:6. It then passes the value in N7:10 through the mask in N7:6. If the two masked values are equal, set output bit O:013/01.

Not Equal to (NEQ)



Description:
Use the NEQ instruction to test whether two values are not equal. Source A and Source B can either be values or addresses that contain values.

Example:



This instruction tells the processor: if the value in N7:5 is not equal to the value in N7:10, set output bit O:013/01.

Chapter 4

Compute Instructions CPT, ACS, ADD, ASN, ATN, AVE, CLR, COS, DIV, LN, LOG, MUL, NEG, SIN, SRT, SQR, STD, SUB, TAN, XPY

Using Compute Instructions

The compute instructions let you evaluate arithmetic operations using an expression or a specific arithmetic instruction. Table 4.A lists the available compute instructions.

Table 4.A
Available Compute Instructions

If You Want to:	Use this Instruction:	Found on Page:
Evaluate an expression	CPT	4-4
Take the arc cosine of a number	ACS *	4-10
Add two values	ADD	4-10
Take the arc sine of a number	ASN *	4-11
Take the arc tangent of a number	ATN *	4-12
Calculate the average for a set of values	AVE *	4-12
Clear an address word (set all bits to zero)	CLR	4-14
Take the cosine of a number	COS *	4-15
Divide two values	DIV	4-15
Take the natural log of a number	LN *	4-16
Take the log of a number	LOG *	4-17
Multiple two values	MUL	4-17
Take the opposite sign of a value	NEG	4-18
Take the sine of a number	SIN *	4-18
Take the square root of a value	SQR	4-19
Sort a set of values into ascending order	SRT *	4-20
Calculate the standard deviation for a set of values	STD *	4-21
Subtract two values	SUB	4-24
Take the tangent of a number	TAN *	4-24
Raise a number to a power	XPY *	4-25

* These instructions are only supported by PLC-5/30, -5/40 and -5/60 processors.

Using Arithmetic Status Flags

The arithmetic status flags are in word 0 bits 0–3 in the processor status file (S). Table 4.B lists the status bits:

Table 4.B
Arithmetic Status Bits

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

Data Types and the Compute Instruction

You can compute values of different data types, such as floating point and integer. If you use a floating-point as the source, use a floating-point as the destination; otherwise, the destination value will be rounded.

You should use BCD and ASCII values for display purposes. If you enter BCD or ASCII values, the processor treats those values as integers.

The parameters you enter are program constants or logical addresses of the values you want.

Using Integer Data Types

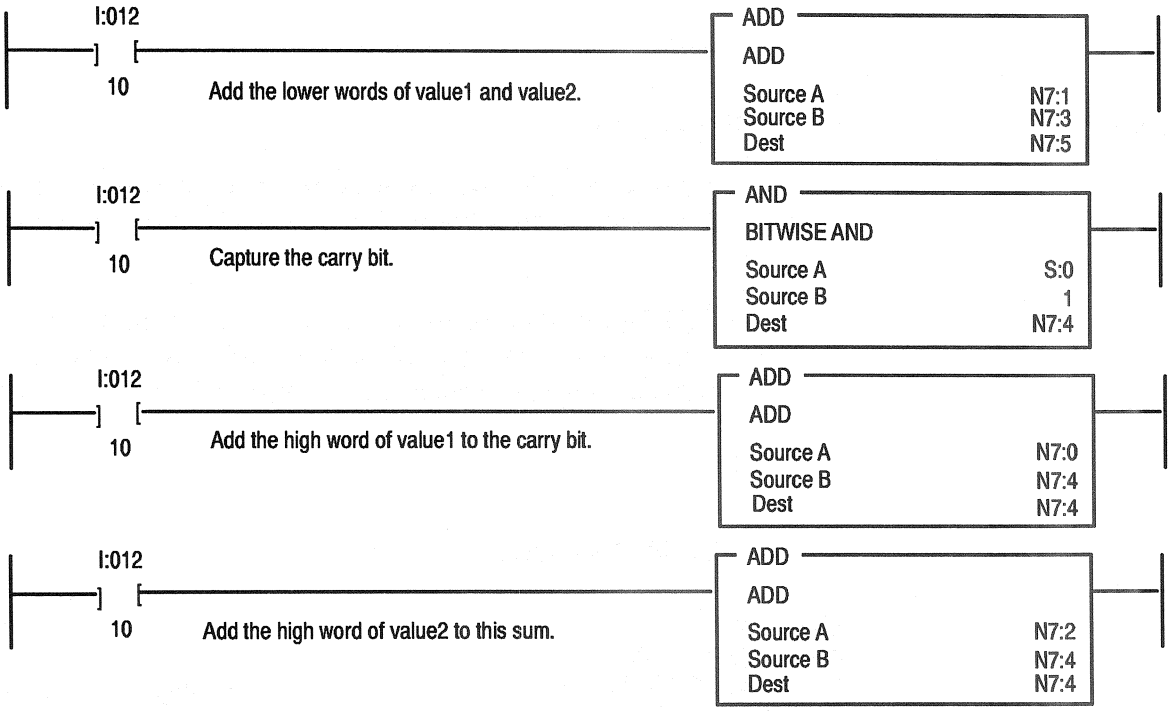
If you use an integer data type with a PLC-5/10, -5/12, -5/15 or -5/25 processor, the processor rounds the final value of a mathematical operation before storing the final result. The processor rounds to the nearest whole number. The processor rounds values of 0.5 through 0.9 up to the next whole number; the processor rounds values of 0.1 through 0.4 down to the closest whole number. If this value is greater than 32767 or less than -32768, the overflow status bit is set.

If you are using a PLC-5/30, -5/40 or -5/60 processor, the processor rounds to the next even number. If this value is greater than 32767 or less than -32768, the processor "wraps" negative (32767, -32768, -32767, -32766, etc.). For example, if you have an ADD instruction with a result greater that 32767, the overflow bit is set, the sign bit is set, and the result is negative: 32767 + 5 = -32764.

Important: If you are using a PLC-5/30, -5/40 or -5/60 and an arithmetic operation generates an overflow, the upper bits are lost, but the lower bits are correct. If you then perform a logical operation on the lower word (AND or OR), you can get the proper result. Also, using the carry bit, you can do multi-word arithmetic (i.e., add two 32-bit words).

For example, if: value1 = N7:0 and N7:1
value2 = N7:2 and N7:3
result = N7:4 and N7:5

and you want to add Value1 to Value2, your ladder program would be:



Using Floating Point Data Types

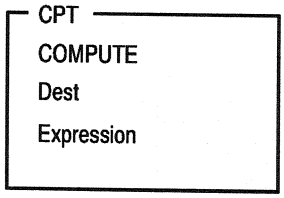
For a PLC-5/30, -5/40 or -5/60 processor, if you use floating point data types and the result is too large or if it is undefined (i.e. natural log of 0), the processor sets the overflow bit.

If the result it too large, a `!+INF!` is displayed; if the result to too small, a `!-INF!` is displayed. If the value is not a number, `!NAN!` is displayed.

Important: If you are using floating point and the number is greater than 32767 or less than -32768, you must use a decimal point. If you do not use a decimal point, the error `INVALID OPERAND` appears.

Also note that when you use complex expressions, if any operand is floating point, the entire expression is evaluated as floating point. Refer to the example in the "Expression Examples" section on 4-7 in this chapter for more information.

Compute (CPT)



The CPT instruction performs copy, arithmetic, logical, and conversion operations.

Description:
The CPT instruction is an output instruction that performs the operations you define in the expression, and writes the result into the destination address.

The CPT instruction also copies data from one address to another, and automatically converts the data type at the source address to the data type you specify in the destination address.

The execution time of a CPT instruction is longer than the execution time of an arithmetic, logic or move instruction (i.e., ADD, AND, MOV, etc.). The CPT instruction also uses more words in your program file.

After each CPT instruction is performed, the arithmetic status bits in the status file of the data table are updated the same as the corresponding arithmetic, logic or move instruction. For example, refer to the description of the ADD instruction to see how the status bits are updated after a CPT (add) instruction is executed.

Entering the CPT Expression

The expression defines the operations you want to perform. You define the expression with operators and addresses or program constants. With PLC-5/30, -5/40 and -5/60 processors, you can enter complex expressions (up to a total of 80 characters, including spaces and parentheses). Table 4.C lists valid operations for an expressions; the following list provides guidelines for writing expressions:

- Operators (symbols) define the operations
- Addresses can be direct or indirect logical address(es) (must be element or bit level)
- With PLC-5/30, -5/40 and -5/60 processors, program constants can be integer or floating-point numbers (if you enter octal values, use a leading &O; if you enter hexadecimal values, use a leading &H)

Table 4.C
Valid Operations for Use in a CPT Expression

Type	Operator	Description	Example Operation
Copy	none	copy from A to B	enter source address in the expression enter destination address in destination
Clear	none	set a value to zero	0 (enter 0 for the expression)

Type	Operator	Description	Example Operation
Arithmetic	+	add	2 + 3 2 + 3 + 7: PLC-5/30, -5/40, -5/60
	-	subtract	12 - 5 (12 - 5) - 7: PLC-5/30, -5/40, -5/60
	*	multiply	5 * 2 6 * (5 * 2): PLC-5/30, -5/40, -5/60
	| (vertical bar)	divide	24 | 6 (24 | 6) *2
	-	negate	- N7:0
	SQR	square root	SQR N7:0
	**	exponential * (x to the power of y)	10**3
	LN	natrual log *	N7:0 LN F8:20
	LOG	log to the base 10 *	N7:2 LOG F8:3
Trigonometric	ACS	arc cosine *	F8:17 ACS F8:18
	ASN	arc sine *	F8:19 ASN F8:20
	ATN	arc tangent *	F8:21 ATN F8:22
	COS	cosine *	F8:13 COS F8:14
	SIN	sine *	F8:11 SIN F8:12
	TAN	tangent *	F8:15 TAN F8:16
Bitwise	AND	bitwise AND	D9:3 AND D10:4
	OR	bitwise OR	D10:4 OR D10:5
	XOR	bitwise exclusive OR	D9:5 XOR D10:4
	NOT	bitwise complement	NOT D9:3
Conversion	FRD	convert from BCD to binary	FRD N7:0
	TOD	convert from binary to BCD	TOD N7:0
	DEG	convert radians to degrees *	F8:7 DEG F8:8
	RAD	convert degrees to radians *	N7:9 RAD F8:10

* Available on PLC-5/30, -5/40, -5/40L, -5/60 and -5/60L processors only.

Determining the Length of an Expression

With PLC-5/30, -5/40, and -5/60 processors, you can enter complex instructions (up to a total of 80 characters, including spaces and parentheses). Depending on the operator, the processor inserts characters before/after the operator in your expression to format the expression for easier interpretation. Use Table 4.D below to determine the number of characters each operator uses in an expression.

Table 4.D
Character Lengths for Operators

This Operation:	Using This Operator:	Uses This Number of Characters:
math binary	+, –, *, |	3
	OR, **	4
	AND, XOR	5
math unary	- (negate)	2
	LN *	3
	FRD, TOD, DEG*, RAD*, SQR, NOT, LOG*, SIN*, COS*, TAN*, ASN*, ACS*, ATN*	4
comparative	=, <, >	3
	<>, <=, >=	4

* Available on PLC-5/30, -5/40 and -5/60 processors only.

Determining the Order of Operation

The operations that you write into the expression are performed by the processor in a prescribed order, not necessarily the order you write them. You can override the order of operation by grouping terms within parentheses, forcing the processor to perform the operation within the parentheses ahead of other operations.

Operations of equal order are performed left to right. The addresses you use must include a bit operator. Table 4.E shows the order of operation.

Table 4.E
Order of Operation for CPT Expressions

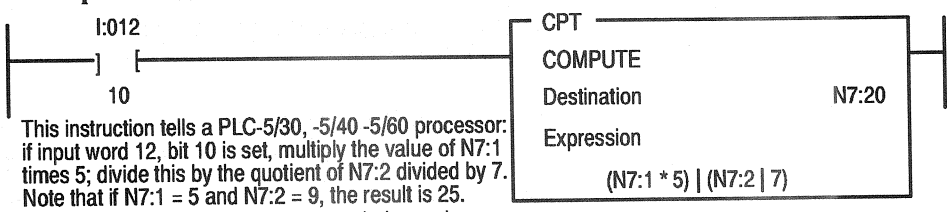
Order	Operation	Description
1	**	exponential (X^Y) PLC-5/30, -5/40 and -5/60 only
2	-	negate
	NOT	bitwise complement
3	*	multiply
	/	divide
4	+	add
	-	subtract
5	AND	bitwise AND
6	XOR	bitwise exclusive OR
7	OR	bitwise OR

Expression Examples

Single Value: The expression SQR (N7:4) with destination N7:20 tells the processor to take the square root of the value stored at N7:4 and store the result at N7:20.

Multiple Values: With PLC-5/30, -5/40 and -5/60 processors, you can also use functions to operate on one or more values in the expression (complex expressions) for compute and compare operations. Complex expressions can be up to 80 characters long (spaces and parentheses are considered characters. For example, you could enter an expression such as:

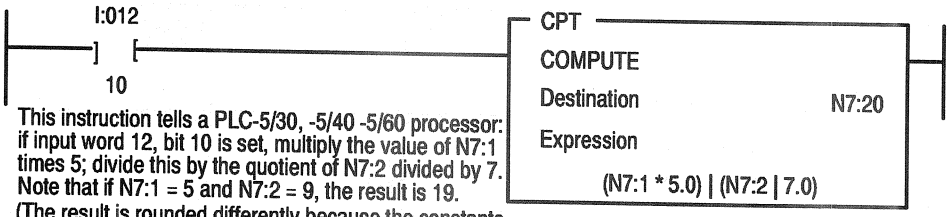
Example:



This instruction tells a PLC-5/30, -5/40 -5/60 processor: if input word 12, bit 10 is set, multiply the value of N7:1 times 5; divide this by the quotient of N7:2 divided by 7. Note that if N7:1 = 5 and N7:2 = 9, the result is 25. (The result is rounded to the nearest whole number because the constants 5 and 7 were specified as whole numbers.)

When you use complex expressions, if any operand is floating point, the entire expression is evaluated as floating point:

Example:



This instruction tells a PLC-5/30, -5/40 -5/60 processor: if input word 12, bit 10 is set, multiply the value of N7:1 times 5; divide this by the quotient of N7:2 divided by 7. Note that if N7:1 = 5 and N7:2 = 9, the result is 19. (The result is rounded differently because the constants 5.0 and 7.0 were specified to 1 decimal place.)

Entering the Destination

Enter a direct or indirect logical address for the destination. The instruction stores the result of the operation in the destination address.

Important: The processor automatically converts the data type specified by the source address to that specified by the destination address. The processor uses BCD only for display or compatibility with PLC-2 family processors. You must program any BCD conversions.

Using CPT Functions

Use functions to operate on one or more values in the expression of a CPT instruction to perform these types of operations:

- Convert from one number format to another
- Manipulate numbers
- Perform trigonometric functions

The instruction performs the function(s) you specify based on a mnemonic. When you enter the expression, enter the mnemonic as the prefix to the address of the value on which you want to operate, or as a prefix to the value itself when entered as a program constant.

Important: Floating-point numbers are 32-bit values. Integers are 16-bit values. The instruction automatically converts the data types found in the expression to the data type specified by the destination address.

WARNING: If the expression or destination addresses require conversion from 32-bit to 16-bit data and the value is too large, the processor sets an overflow bit in S:0/1 and sets a minor fault (S10:14). The resulting erroneous value could lead to a dangerous situation. Monitor this bit in your ladder program.

Table 4.F, Table 4.G and Table 4.H list the CPT functions you can use.

Table 4.F
CPT Functions for Number Conversion

Mnemonic	Title	Description
RAD *	radians	Converts from degrees to radians
DEG *	degrees	Converts from radians to degrees
TOD	to BCD	Converts from integer to BCD (supports 4-digit BCD numbers)
FRD	from BCD	Converts from BCD to integer (supports 4-digit BCD numbers)

* Available on PLC-5/30, -5/40 and -5/60 processors only.

You can use the above CPT conversion functions within expressions or as stand-alone instructions; see chapter 6 in this module.

Table 4.G
CPT Functions for Number Manipulation (accurate up to 6 significant digits)

Mnemonic	Title	Description
SQR	square root	Takes the square root of the number
LOG *	-	Log to the base 10
LN *	-	Natural log

* Available on PLC-5/30, -5/40 and -5/60 processors only.

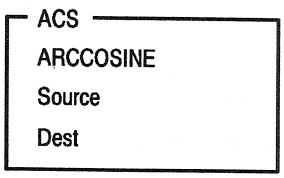
Table 4.H
CPT Trigonometric Functions (manipulated in radians, accurate up to 6 significant digits)

Mnemonic	Title
SIN *	sine
COS *	cosine
TAN *	tangent
ASN *	inverse sine
ACS *	inverse cosine
ATN *	inverse tangent

* Available on PLC-5/30, -5/40 and -5/60 processors only.

You can use the above CPT arithmetic functions within expressions or as stand-alone instructions; see the individual instructions described in this chapter.

Arc Cosine (ACS) (PLC-5/30, -5/40, -5/60 only)



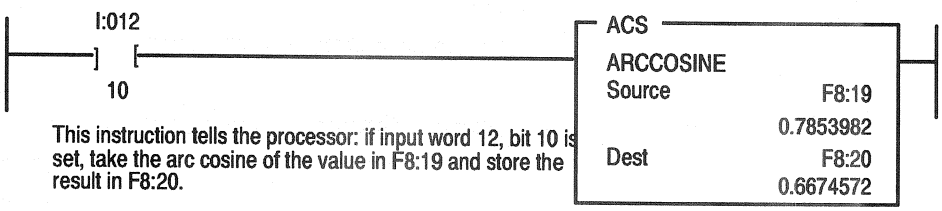
Description:
Use the ACS instruction to take the arc cosine of a number (Source A in radians) and store the result (in radians) in the Destination.

The value in the Source must be greater than or equal to -1 and less than or equal to 1. If it is not in this range, the processor returns a !NAN! result in the Destination. The resulting value in the Destination is always greater than or equal to 0 and less than or equal to π (where $\pi = 3.141592$).

Table 4.I
Updating Arithmetic Status Flags for an ACS Instruction

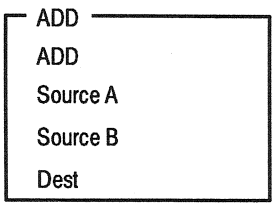
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the arc cosine of the value in F8:19 and store the result in F8:20.

Addition (ADD)



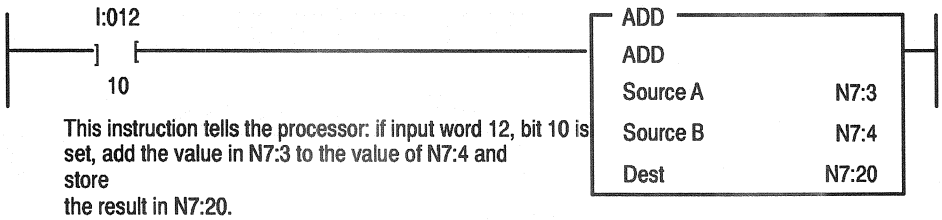
Description:
Use the ADD instruction to add one value (Source A) to another value (Source B) and places the result in the destination. Source A and Source B can either be values or addresses that contain values.

Important: The compute instructions execute for each scan as long as the rung is true; if you only want values computed once, include the ONS command (see chapter 13).

Table 4.J
Updating Arithmetic Status Flags for an ADD Instruction

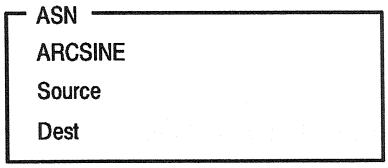
With this Bit:	The Processor:
Carry (C)	sets if carry generated; otherwise resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, add the value in N7:3 to the value of N7:4 and store the result in N7:20.

Arc Sine (ASN) (PLC-5/30, -5/40, -5/60 only)



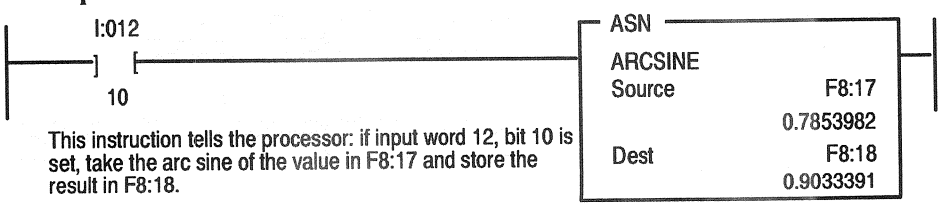
Description:
Use the ASN instruction to take the arc sine of a number (Source A in radians) and store the result (in radians) in the Destination.

The value in the Source must be greater than or equal to -1 and less than or equal to 1. If it is not in this range, the processor returns a !NAN! result in the Destination. The resulting value in the Destination is always greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$ (where $\pi = 3.141592$).

Table 4.K
Updating Arithmetic Status Flags for an ASN Instruction

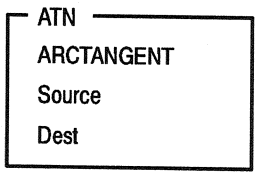
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the arc sine of the value in F8:17 and store the result in F8:18.

Arc Tangent (ATN) (PLC-5/30, -5/40, -5/60 only)

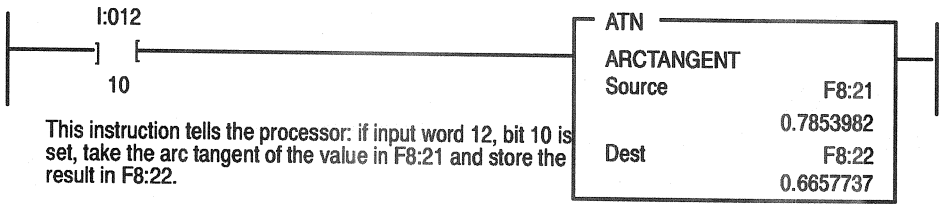


Description:
Use the ATN instruction to take the arc tangent of a number (Source A in radians) and store the result (in radians) in the Destination. The resulting value in the Destination is always greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$ (where $\pi = 3.141592$).

Table 4.L
Updating Arithmetic Status Flags for an ATN Instruction

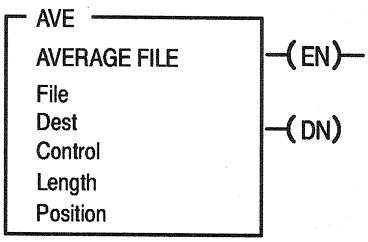
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the arc tangent of the value in F8:21 and store the result in F8:22.

Average File (AVE) (PLC-5/30, -5/40, -5/60 only)



Description:
The AVE instruction calculates the average of a set of values. When the rung goes from false to true, the value at the current position is added to the next value, which is added to the next value, and so on.

Each time another value is added, the position field and the status word (S:24) is incremented. The final sum is divided by the number of values added and the result is stored in the destination.

Table 4.M
Updating Arithmetic Status Flags for an AVE Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

An overflow can occur if:

- the intermediate sum exceeds the maximum floating point value
- the destination is an integer address and the final value is greater than 32,767 or less than -32768

If an overflow occurs, the processor aborts the calculation and the .ER bit is set and the destination remains unchanged. The position points to the element that caused the overflow. When you clear the .ER bit, the position resets to 0 and the average is recalculated.

Important: Use the RES instruction to clear the status flags.

Entering Parameters

To program the AVE instruction, you must provide the processor with the following:

- **File** is the address that contains the first value to be added. This address can be floating point or integer.
- **Destination** is the address where the result of the instruction is stored. This address can be floating point or integer.
- **Control** is the address of the control structure in the control area (R) of processor memory. The processor uses this information to run the instruction.
- **Length** is the number of words in the file (0–1000).
- **Position** points to the word that the instruction is currently using.

Using Status Bits

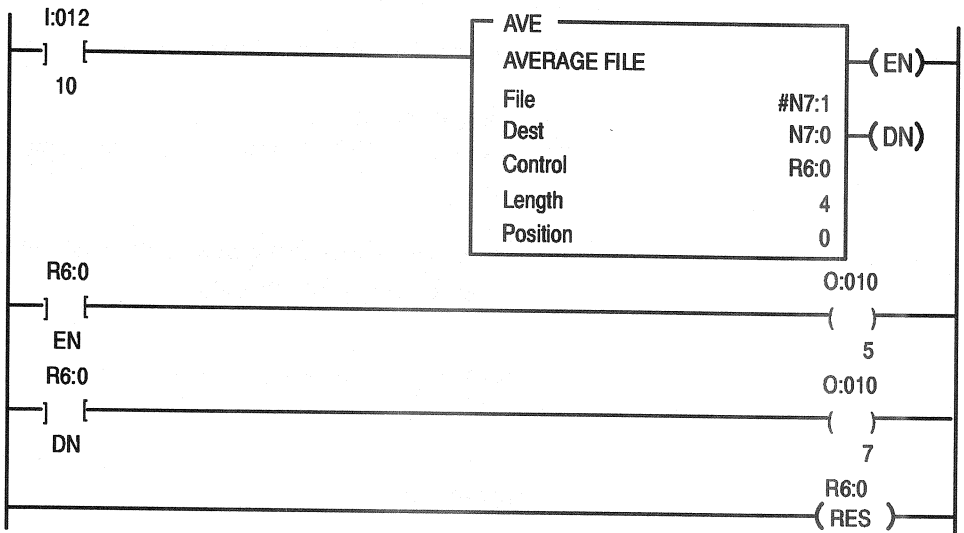
To use the AVE instruction correctly, the ladder program must examine status bits in the control structure. You address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition.
Done .DN (bit 13)	after the instruction finishes operating. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the operation generates an overflow. The instruction stops until the ladder program resets the .ER bit.

Important: The AVE instruction calculates the average using floating point regardless of the type specified for the file or destination parameters.

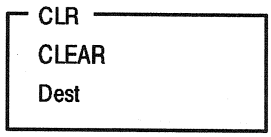
WARNING: The AVE instruction manipulates the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise, unpredictable machine operation could occur with possible damage to equipment and/or personnel.

Example:



This instruction tells the processor: if input word 12, bit 10 is set, the AVE instruction is enabled. The values in N7:1, N7:2, N7:3 and N7:4 are added and divided by 4, and the result is stored in N7:0. When the calculation is complete, output word 10, bit 7 is set. The RES instruction then resets the status bits of the control file.

Clear (CLR)

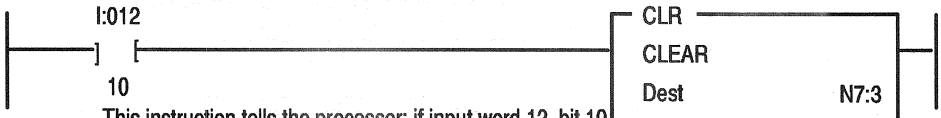


Description:
Use the CLR instruction to set all the bits of a word to zero. The destination must be a word address.

Table 4.N
Updating Arithmetic Status Flags for a CLR Instruction

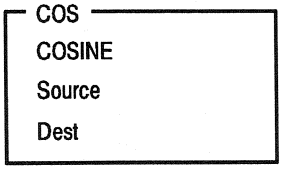
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	always sets
Sign (S)	always resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, set all of the bits in N7:3 to zero.

Cosine (COS) (PLC-5/30, -5/40, -5/60 only)



Description:
Use the COS instruction to take the cosine of a number (Source A in radians) and store the result in the Destination.

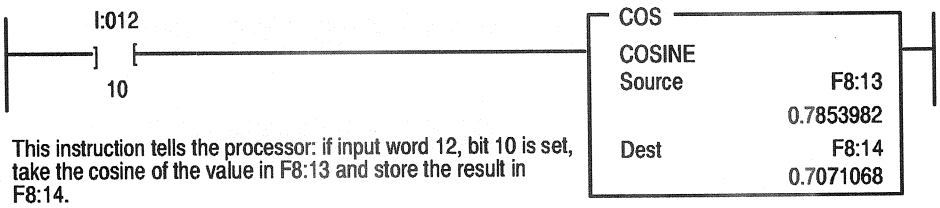
The value in the Source must be greater than or equal to -205887.4 and less than or equal to 205887.4. If it is not in this range, the processor returns a !INF! result in the Destination. The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

Important: For greatest accuracy, the source data should be greater than or equal to -2π and less than or equal to 2π.

Table 4.0
Updating Arithmetic Status Flags for an COS Instruction

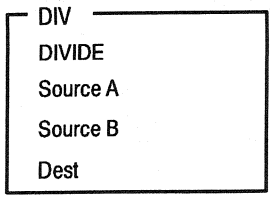
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the cosine of the value in F8:13 and store the result in F8:14.

Divide (DIV)



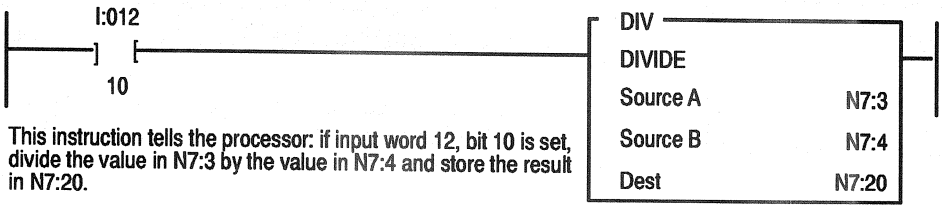
Description:
Use the DIV instruction to divide one value (Source A) by another value (Source B) and place the result in the destination. Source A and Source B can either be values or addresses that contain values.

Important: The compute instructions execute for each scan as long as the rung is true; if you only want values computed once, include the ONS command (see the Program Control instructions in this manual).

Table 4.P
Updating Arithmetic Status Flags for a DIV Instruction

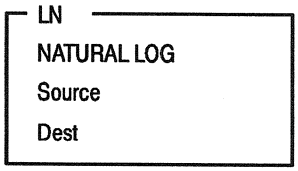
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if division by zero or if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets; undefined if overflow is set
Sign (S)	sets if result is negative; otherwise resets; undefined if overflow is set

Example:



This instruction tells the processor: if input word 12, bit 10 is set, divide the value in N7:3 by the value in N7:4 and store the result in N7:20.

Natural Log (LN) (PLC-5/30, -5/40, -5/60 only)



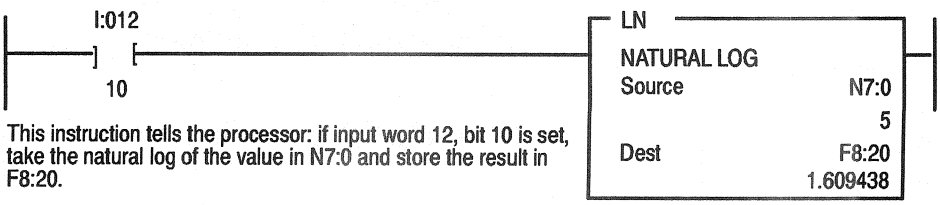
Description:
Use the LN instruction to take the natural log of the value in Source A and store the result in the Destination.

If the value in the Source is equal to 0, the result in the Destination will be !-INF!; if the value in the Source is less than 0, the result in the Destination will be !NAN!. The resulting value in the Destination is always greater than or equal to -87.33655 and less than or equal to 88.72284.

Table 4.Q
Updating Arithmetic Status Flags for an LN Instruction

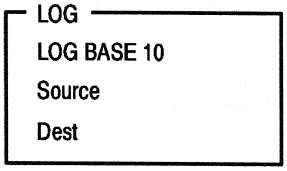
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the natural log of the value in N7:0 and store the result in F8:20.

Log to the Base 10 (LOG) (PLC-5/30, -5/40, -5/60 only)



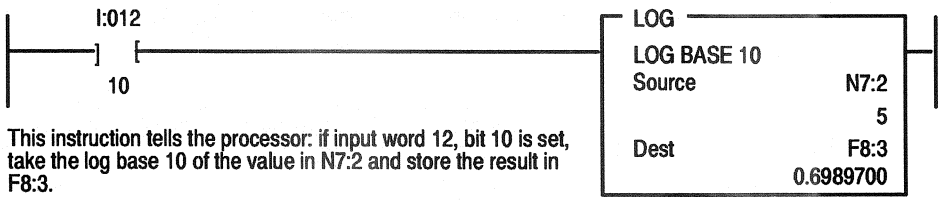
Description:
Use the LOG instruction to take the log base 10 of the value in Source A and store the result in the Destination.

If the value in the Source is equal to 0, the result in the Destination will be !-INF!; if the value in the Source is less than 0, the result in the Destination will be !NAN!. The resulting value in the Destination is always greater than or equal to -37.92978 and less than or equal to 38.53184.

Table 4.R
Updating Arithmetic Status Flags for an LOG Instruction

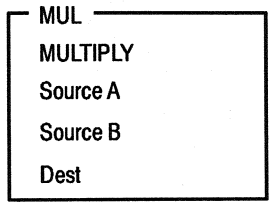
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the log base 10 of the value in N7:2 and store the result in F8:3.

Multiply (MUL)

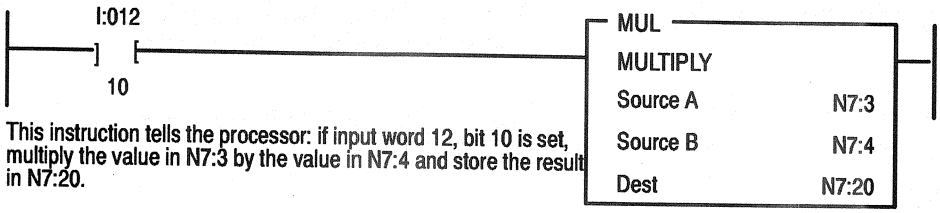


Description:
Use the MUL instruction to multiply one value (Source A) by another value (Source B) and place the result in the destination. Source A and Source B can either be values or addresses that contain values.

Table 4.S
Updating Arithmetic Status Flags for a MUL Instruction

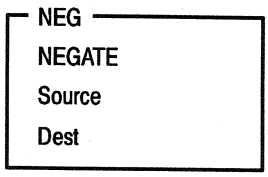
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, multiply the value in N7:3 by the value in N7:4 and store the result in N7:20.

Negate (NEG)



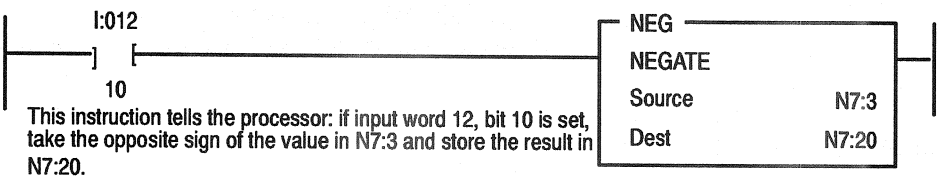
Description:
Use the NEG instruction to change the sign of a value. If you negate a negative value, the result is positive; if you negate a positive value, the result is negative.

Important: The compute instructions execute for each scan as long as the rung is true; if you only want values computed once, include the ONS command (see chapter 13 in this manual).

Table 4.T
Updating Arithmetic Status Flags for a NEG Instruction

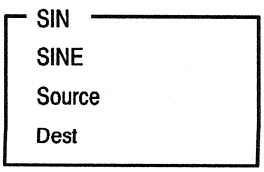
With this Bit:	The Processor:
Carry (C)	sets if the operation generates a carry
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the opposite sign of the value in N7:3 and store the result in N7:20.

Sine (SIN) (PLC-5/30, -5/40, -5/60 only)



Description:
Use the SIN instruction to take the sine of a number (Source A in radians) and store the result in the Destination.

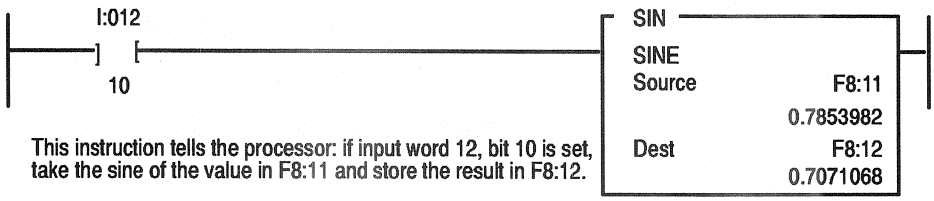
The value in the Source must be greater than or equal to -205887.4 and less than or equal to 205887.4. If it is not in this range, the processor returns a !INF! result in the Destination. The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

Important: For greatest accuracy, the source data should be greater than or equal to -2π and less than or equal to 2π.

Table 4.U
Updating Arithmetic Status Flags for an SIN Instruction

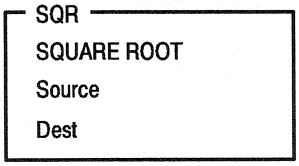
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the sine of the value in F8:11 and store the result in F8:12.

Square Root (SQR)



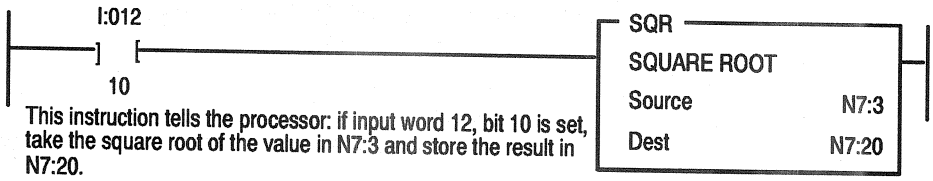
Description:
Use the SQR instruction to take the square root of a value and store the result in the destination. The source can be a value or an address that contains a value.

Important: The compute instructions execute for each scan as long as the rung is true; if you only want values computed once, include the ONS command (see chapter 13 in this manual).

Table 4.V
Updating Arithmetic Status Flags for a SQR Instruction

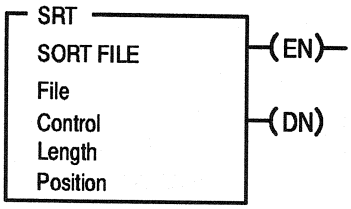
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated during floating-point to integer conversion; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	always resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the square root of the value in N7:3 and store the result in N7:20.

Sort File (SRT) (PLC-5/30, -5/40, -5/60 only)



Description:
The SRT instruction sorts a set of values into ascending order. This instruction is executed on a false-to-true transition.

Important: Make sure the file length value you specify in the instruction does not cause the indexed address to exceed the file bounds. The processor does not check this unless you exceed the data file area of memory. If the indexed address exceeds the data file area, the processor initiates a run-time error and sets a major fault. The processor does not check to see whether the indexed address crosses file types, such as N7 to F8.

Entering Parameters
To program the SRT instruction, you must provide the processor with the following:

- **File** is the address that contains the first word to be sorted. This address can be floating point or integer.
- **Control** is the address of the control structure in the control area (R) of processor memory. The processor uses this information to run the instruction.
- **Length** is the number of words in the file (0–1000).
- **Position** points to the element that the instruction is currently using.

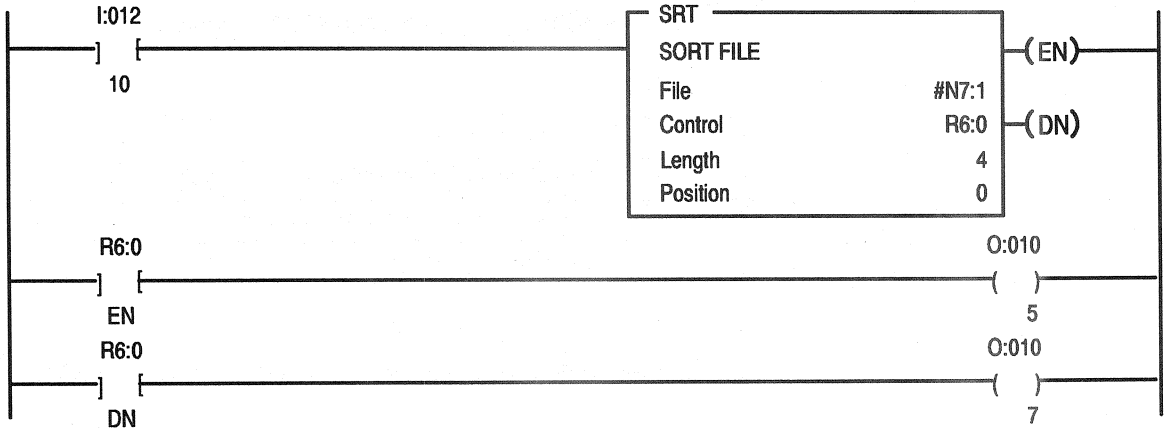
Using Status Bits

To use the SRT instruction correctly, the ladder program must examine status bits in the control structure. You address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition.
Done .DN (bit 13)	after the instruction finishes operating. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the length value is less than or equal to zero or when the position value is less than zero.

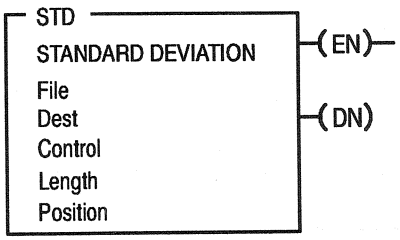
WARNING: The SRT instruction manipulates the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise, unpredictable machine operation could occur with possible damage to equipment and/or personnel.

Example:



This instruction tells the processor: if input word 12, bit 10 is set, the SRT instruction is enabled. The elements in N7:1, N7:2, N7:3 and N7:4 are sorted into ascending order. When the calculation is complete, output word 10, bit 7 is set.

Standard Deviation (STD) (PLC-5/30, -5/40, -5/60 only)



Description:
The STD instruction calculates the standard deviation of a set of values and stores the result in the destination. This instruction is executed on a false-to-true transition.

The standard deviation is calculated according to the following formula:

$$\text{Standard Deviation} = \sqrt{\frac{[N * SqSum] - Sum^2}{N * (N - 1)}}$$

Where:

Sum – sum of the values in the file

N – number of values in the file

SqSum – sum of all values in the file squared ($a^2 + b^2$ + etc.)

Important: Make sure the file length value you specify in the instruction does not cause the indexed address to exceed the file bounds. The processor does not check this unless you use an indexed indirect address or exceed the data file area of memory. If the indexed address exceeds the data file area, the processor initiates a run-time error and sets a major fault. The processor does not check to see whether the indexed address crosses file types, such as N7 to F8.

Table 4.W
Updating Arithmetic Status Flags for an STD Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	always resets

There are two ways an overflow can occur:

- the intermediate sum exceeds the maximum floating point value (floating point values are: $\pm 1.1754944^{e-38}$ to $\pm 3.4028237^{e+38}$)
- the destination is an integer address and the final value is greater than 32,767

If an overflow occurs, the processor aborts the calculation and the .ER bit is set and the destination remains unchanged. The position points to the element that caused the overflow. When you clear the .ER bit, the position resets to 0 and the standard deviation is recalculated.

Important: Use the RES instruction to clear the status bits.

Entering Parameters

To program the STD instruction, you must provide the processor with the following:

Parameter:	Definition:
file	the address that contains the first value to be calculated. This address can be floating point or integer.
destination	the address where the result of the instruction is stored. This address can be floating point or integer.
control	the address of the control structure in the control area (R) of processor memory. The processor uses this information to run the instruction.
length	the number of words in the file (0–1000).
position	points to the element that the instruction is currently using.

Using Status Bits

To use the STD instruction correctly, the ladder program must examine status bits in the control structure. You address these bits by mnemonic.

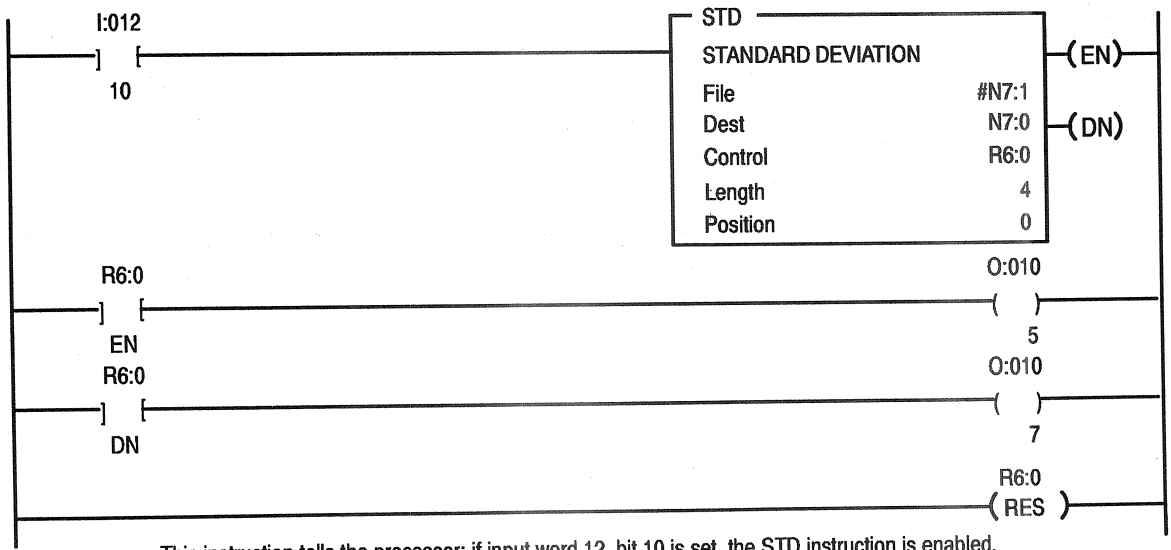
This Bit:	Is Set:
Enable .EN (bit 15)	on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition.
Done .DN (bit 13)	after the instruction finishes operating. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the operation generates an overflow. The instruction stops until the ladder program resets the .ER bit.

Important: The STD instruction calculates the standard deviation using floating point regardless of the type specified for the file or destination parameters.



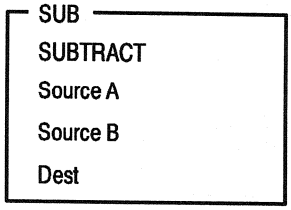
WARNING: The STD instruction manipulates the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise, unpredictable machine operation could occur with possible damage to equipment and/or personnel.

Example:



This instruction tells the processor: if input word 12, bit 10 is set, the STD instruction is enabled. The elements in N7:1, N7:2, N7:3 and N7:4 are used to calculate the standard deviation. When the calculation is complete, output word 10, bit 7 is set. The RES instruction then resets the status bits.

Subtract (SUB)



Description:
Use the SUB instruction to subtract one value (Source B) from another value (Source A) and place the result in the destination. Source A and Source B can either be values or addresses that contain values.

Important: The compute instructions execute for each scan as long as the rung is true; if you ony want values computed once, include the ONS command (see chapter 13).

Table 4.X
Updating Arithmetic Status Flagss for a SUB Instruction

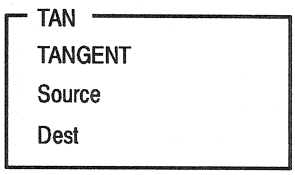
With this Bit:	The Processor:
Carry (C)	sets if borrow generated; otherwise resets
Overflow (V)	sets if underflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, subtract the value in N7:3 by the value in N7:4 and store the result in N7:20.

Tangent (TAN) (PLC-5/30, -5/40, -5/60 only)



Description:
Use the TAN instruction to take the tangent of a number (Source A in radians) and store the result in the Destination.

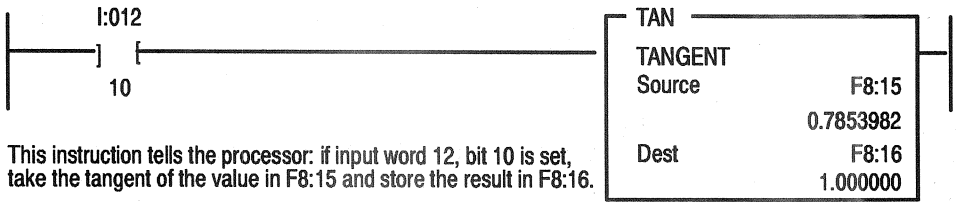
The value in the Source must be greater than or equal to -102943.7 and less than or equal to 102943.7. If it is not in this range, the processor returns a !INF! result in the Destination. The resulting value in the Destination is always a real number.

Important: For greatest accuracy, the source data should be greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$.

Table 4.Y
Updating Arithmetic Status Flags for an TAN Instruction

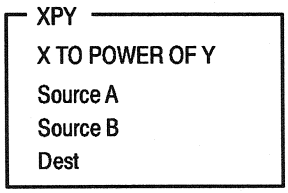
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the tangent of the value in F8:15 and store the result in F8:16.

X to the Power of Y (XPY) (PLC-5/30, -5/40, -5/60 only)



Description:

Use the XPY instruction to raise a value (Source A) to a power (Source B) and store the result in the Destination. If the value in Source A is negative, the exponent (Source B) should be an integer value; if the exponent is not an integer (for example, if it is a floating point value), the overflow bit is set and the absolute value of the base is used in the calculation.

The XPY instruction uses the following algorithm:

```
XPY = 10 ** (Y * log (X))
```

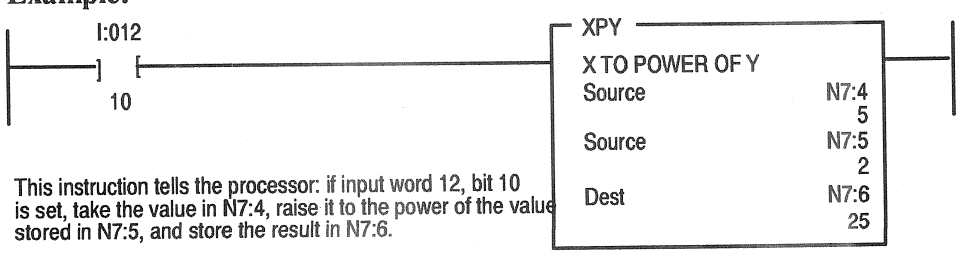
If any of the intermediate operations in this algorithm produce an overflow, the arithmetic minor fault bit is set (S:10/14). The arithmetic status flag bit is set only if the final result is an overflow.

Important: Note that x^0 is equal to 1; 0^x is equal to 0. For floating point numbers, 0^0 is equal to !NAN! and for integers, 0^0 is equal to -1.

Table 4.Z
Updating Arithmetic Status Flags for an XPY Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, take the value in N7:4, raise it to the power of the value stored in N7:5, and store the result in N7:6.

Chapter 5

Logical Instructions AND, NOT, OR, XOR

Using Logical Instructions

These instructions (Table 5.A) let you perform logical operations.

Table 5.A
Available Logical Instructions

If You Want to:	Use this Instruction:	Found on Page:
Perform an AND operation	AND	5-2
Perform a NOT operation	NOT	5-3
Perform an OR operation	OR	5-4
Perform an XOR operation	XOR	5-5

The parameters you enter are program constants or direct logical addresses.

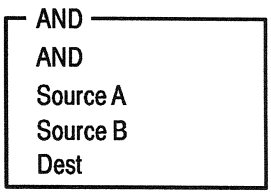
Using Arithmetic Status Flags

The arithmetic status bits are in word 0 bits 0–3 in the processor status file (S). Table 5.B lists the status flags:

Table 5.B
Arithemtic Status Flags

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

AND Operation (AND)



Description:
Use the AND instruction to perform an AND operation using the bits in the two source addresses.

Table 5.C shows how the processor evaluates an AND operation.

Table 5.C
Truth Table for an AND Operation

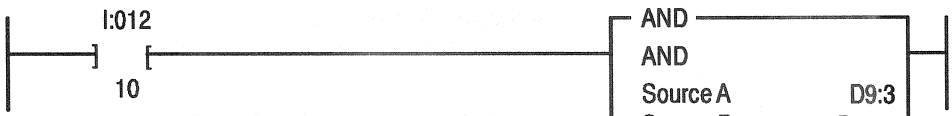
Source A	Source B	Result
0	0	0
1	0	0
0	1	0
1	1	1

Table 5.D describes how the processor updates the arithmetic status flags.

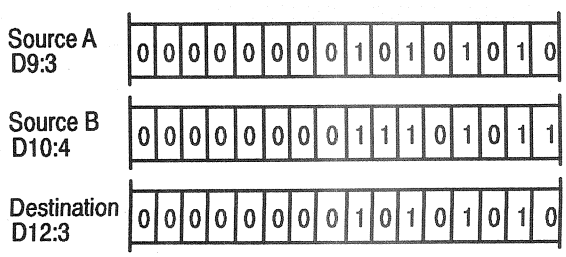
Table 5.D
Updating Arithmetic Status Flags for an AND Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

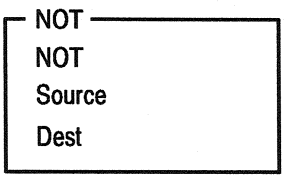
Example:



This instruction tells the processor: if input word 12, bit 10 is set, the processor performs an AND operation on D9:3 and D10:4 and stores the result in D12:3.



NOT Operation (NOT)



Description:
Use the NOT instruction to perform an NOT operation using the bits in the source address. This operation is also know as a bit inversion.

Important: The NOT instruction is not available on PLC-5/15 Series A processors.

Table 5.E shows how the processor evaluates a NOT operation.

Table 5.E
Truth Table for a NOT Operation

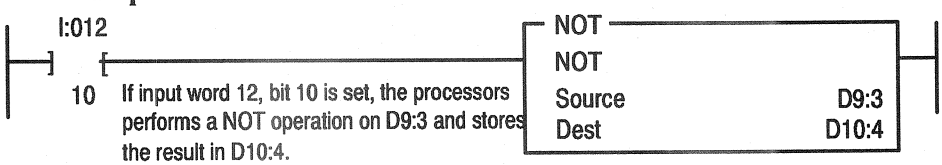
Source	Result
0	1
1	0

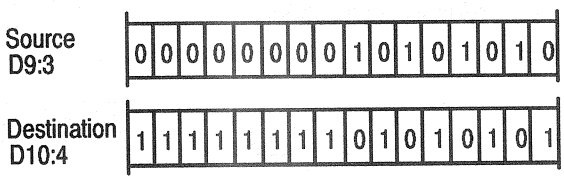
Table 5.F describes how the processor updates the arithmetic status flags.

Table 5.F
Updating Arithmetic Status Flags for a NOT Instruction

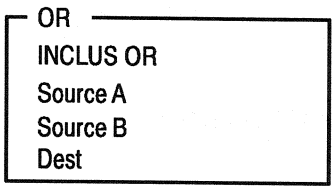
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

Example:





OR Operation (OR)



Description:
Use the OR instruction to perform an OR operation using the bits in the two source addresses.

Table 5.G shows how the processor evaluates an OR operation.

Table 5.G
Truth Table for an OR Operation

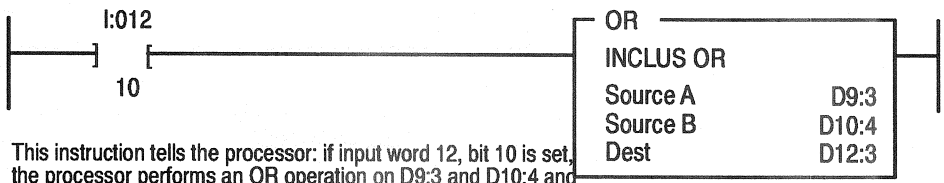
Source A	Source B	Result
0	0	0
1	0	1
0	1	1
1	1	1

Table 5.H describes how the processor updates the arithmetic status flags.

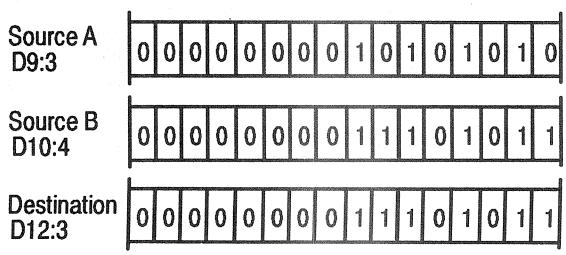
Table 5.H
Updating Arithmetic Status Flags for an OR Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

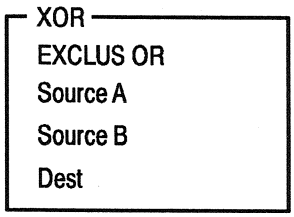
Example:



This instruction tells the processor: if input word 12, bit 10 is set, the processor performs an OR operation on D9:3 and D10:4 and stores the result in D12:3.



Exclusive OR Operation (XOR)



Description:
Use the XOR instruction to perform an exclusive OR operation using the bits in the two source addresses.

Table 5.I shows how the processor evaluates an XOR operation:

Table 5.I
Truth Table for an XOR Operation

Source A	Source B	Result
0	0	0
1	0	1
0	1	1
1	1	0

Table 5.J describes how the processor updates the arithmetic status bits.

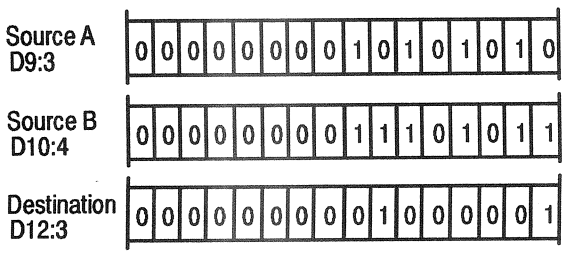
Table 5.J
Updating Arithmetic Status Bits for an XOR Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, the processor performs an XOR operation on D9:3 and D10:4 and stores the result in D12:3.



Chapter 6

Conversion Instructions FRD and TOD, DEG and RAD

Using the Conversion Instructions

The conversion instructions let you convert integer to BCD and convert BCD to integer (using TOD and FRD). For example, use TOD and FRD for signals to/from BCD I/O devices, for display purposes, or for number compatibility with PLC–2 family processors. You can also convert radians to degrees and degrees to radians (using DEG and RAD). For example, you can use DEG and RAD with the trigonometric instructions (described in chapter 4).

Table 6.A lists the available conversion instructions.

Table 6.A
Available Conversion Instructions

If You Want to:	Use this Instruction:	Found on Page:
Convert from integer to BCD	TOD	6-2
Convert from BCD to integer	FRD	6-2
Convert radians to degrees	DEG *	6-3
Convert degrees to radians	RAD *	6-4

* These instructions are only supported by PLC-5/30, -5/40 and -5/60 processors.

The parameters you enter are program constants or logical addresses of the values you want.

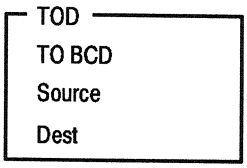
Using Arithmetic Status Flags

The arithmetic status flags are in word 0 bits 0–3 in the processor status file (S2). Table 6.B lists the status flags:

Table 6.B
Arithmetic Status Flags

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

Convert to BCD (TOD)



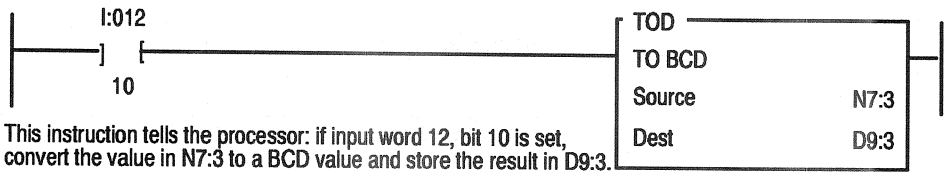
Description:

Use the TOD instruction to convert an integer value to a BCD value. If the integer value is greater than 9999, the processor stores 9999 and sets the overflow bit. If the integer value is negative, the processor stores 0 in the destination and sets the overflow and zero status bits.

Table 6.C
Updating Arithmetic Status Flags for a TOD Instruction

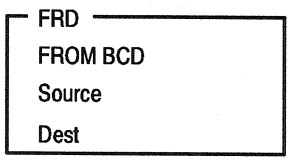
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if integer value is outside the range 0–9999; otherwise resets
Zero (Z)	sets if destination value is negative or zero; otherwise resets
Sign (S)	always resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, convert the value in N7:3 to a BCD value and store the result in D9:3.

Convert from BCD (FRD)



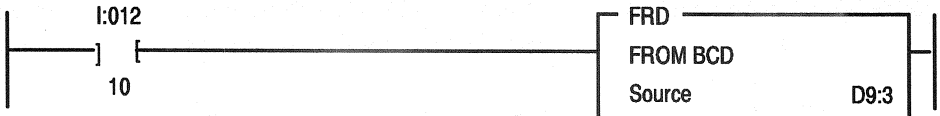
Description:

Use the FRD instruction to convert a BCD value to an integer value. You should convert BCD values to integer values before you manipulate those values with ladder logic because the processor treats BCD values as integer values. The actual BCD value may be lost or distorted.

Table 6.D
Updating Arithmetic Status Flags for a TOD Instruction

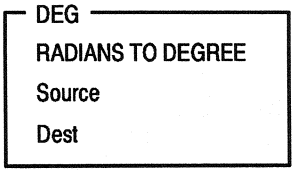
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if destination value is zero; otherwise resets
Sign (S)	always resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, convert the value in D9:3 to an integer value and store the result in N7:3.

Degree (DEG) (PLC-5/30, -5/40, -5/60 only)



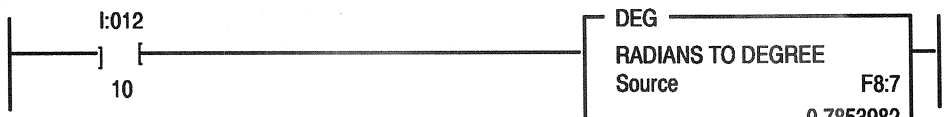
Description:

Use the DEG instruction to convert radians (the value in Source A) to degrees and store the result in the Destination (Source times 180/π).

Table 6.E
Updating Arithmetic Status Flags for an DEG Instruction

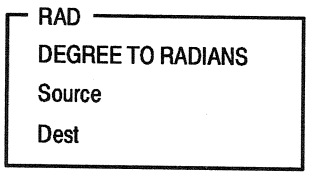
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, convert the value in F8:7 to degrees and store the result in F8:8.

Radian (RAD) (PLC-5/30, -5/40, -5/60 only)



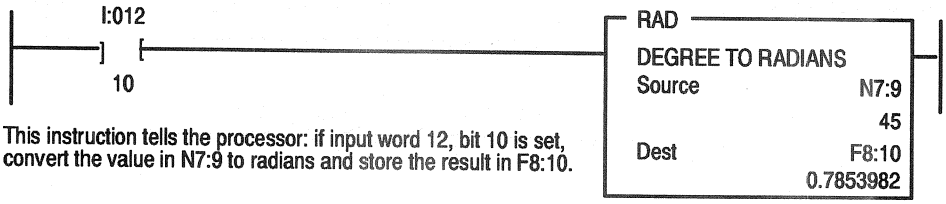
Description:

Use the RAD instruction to convert degrees (the value in Source A) to radians and store the result in the Destination (Source times $\pi/180$).

Table 6.F
Updating Arithmetic Status Flags for an RAD Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



This instruction tells the processor: if input word 12, bit 10 is set, convert the value in N7:9 to radians and store the result in F8:10.

Chapter 7

Bit Modify and Move Instructions BTD, MOV, MVM

Using Bit Modify and Move Instructions

The bit modify and move instructions let you modify and move bits. Table 7.A lists the available move instructions.

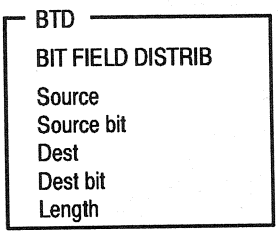
Table 7.A
Available Bit Modify and Move Instructions

If You Want to:	Use This Instruction:	Found on Page:
Move bits within a word or between words	BTD	7-1
Copy the value in one word to another word	MOV	7-3
Copy the desired part of a 16-bit value by masking the rest of the value with a mask	MVM	7-3

These instructions operate on 16-bit integer, binary or floating point numbers to move or copy bits between words. The MVM instruction uses a mask to either pass or block source data bits. A mask passes data when the mask bits are set (1); a mask blocks data when the mask bits are reset (0). The mask must be the same word size as the source and destination.

When rounding floating point numbers during a move to an integer word, the processor does not correctly round numbers less than –1.

Bit Distribute (BTD)



Description:
The BTD instruction is an output instruction that moves up to 16 bits of data within a word or between words. The source remains unchanged. The instruction writes over the destination with the specified bits. If the length of the bit field extends beyond the destination word, the processor does not save the overflow bits. These overflow bits are lost; they do not wrap into the next word.

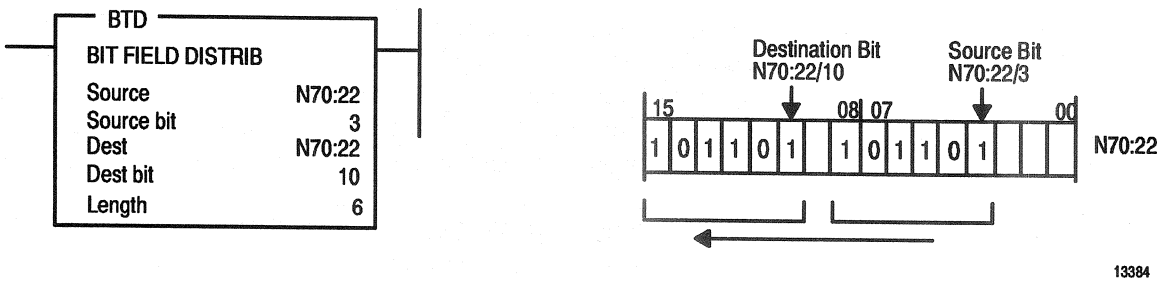
Entering Parameters

To program the BTD instruction, you must provide the processor with the following:

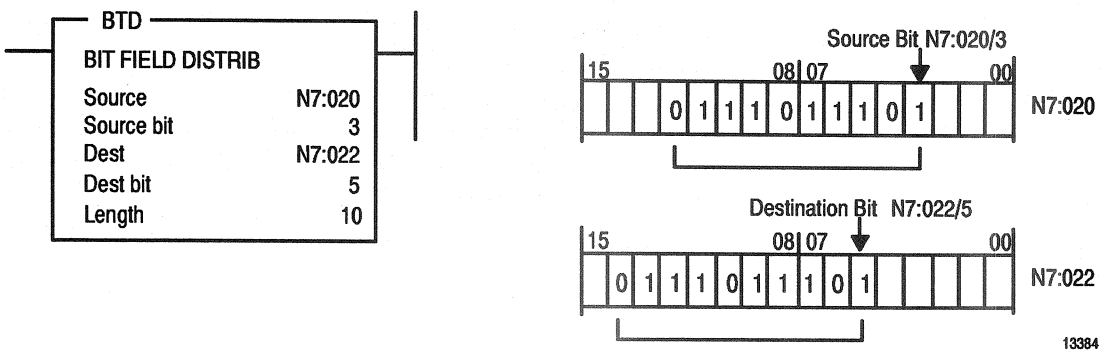
Parameter:	Definition:
source	the address of the source word in binary or integer. The source remains unchanged.
source bits	the number of the bit (lowest bit number) in the source word from which to start the move.
destination	the address of the destination word in a binary or integer file. The instruction writes over any data already stored at the destination.
destination bits	the number of the bit (lowest bit number) in the destination word where the processor starts copying the bits from the source word.
length	the number of bits to be moved

On each scan, when the rung that contains the BTD instruction is true, the processor moves the bit field from the source word to the destination word. To move data within a word, enter the same address for the source and destination.

BTD Example: Moving Bits Within a Word

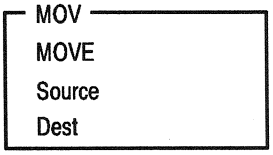


BTD Example: Moving Bits Between Words



Important: Bits are lost if they extend beyond the end of the destination word; the bits are **not** wrapped to the next higher word.

Move (MOV)

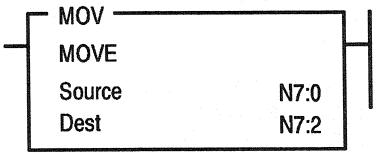


Description:
The MOV instruction is an output instruction that copies a value from a source address to a destination address. As long as the rung remains true, the instruction moves the data each scan.

Table 7.B
Updating Arithmetic Status Flags for a MOV Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated during floating point -to-integer conversion; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

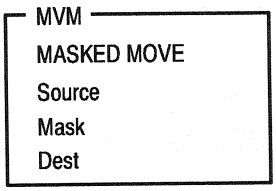
Example:



Entering Parameters:
To program this instruction, you must provide the processor with the following:

Parameter:	Definition:
source	is a program constant or data address from which the instruction reads an image of the value. You can also use a symbol, as long as the symbol name is more than 1 character. The source remains unchanged.
destination	the data address to which the instruction writes the result of the operation. The instruction writes over any data stored at the destination.

Masked Move (MVM)



Description:
The MVM instruction is an output instruction that copies a value from a source address to a destination address, and allows portions of the data to be masked. As long as the rung remains true, the instructions moves the data each scan.

When the rung containing this instruction goes true, data at the source address passes through the mask to the destination address. As long as the rung remains true, the instruction copies the data each scan.

The source and destination data types should be the same because the instruction does not perform a conversion.

You can use the MVM instruction to copy I/O image, binary, or integer values. For example, use MVM to extract bit data such as status or control bits from an element that contains bit and word data.

Table 7.C
Updating Arithmetic Status Flags for a MVM Instruction

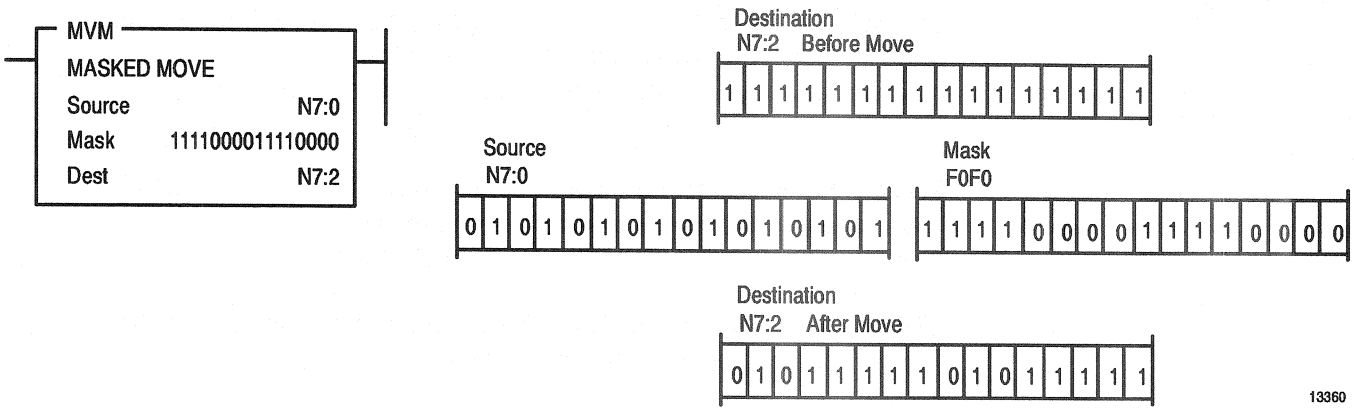
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Entering Parameters:

To program this instruction, you must provide the processor with the following:

Parameter:	Definition:
source	a program constant or data address from which the instruction reads an image of the value. The source remains unchanged.
mask	an address or hexadecimal value that specifies which bits to pass or block. You must set (1) mask bits to move data. Moved data overwrites destination data. Bits at the destination that correspond to zeros in the mask are not altered. If you want the ladder program to change the mask value, store the mask at a data address. When you enter a value in this field, make sure that you include the data type, file number and word number. For example, type `B100:0`. Otherwise, enter a hexadecimal value for a constant mask value. For example, type `F800`.
destination	the data address to which the instruction writes the result of the operation. The instruction writes over any data stored at the destination.

MVM Example:



Chapter 8

File Instruction Concepts

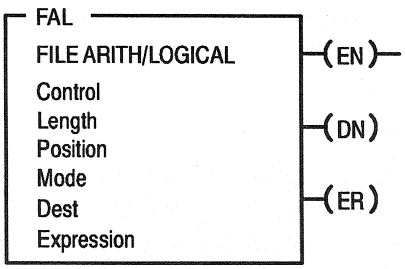
Concepts of File Operation

This chapter presents concepts of file operation for the File Arithmetic and Logical (FAL) and File Search and Compare (FSC) instructions.

The FAL instruction performs arithmetic and logical operations on multiple words. The FSC instruction performs comparison operations on multiple words. For specific information about the FAL or FSC instruction, see the next chapter.

Entering Parameters

When you program a FAL or FSC instruction, the instruction uses the following format:



You need to provide the processor with the following information when you enter a file instruction:

- **Control** is the address of the control structure in the control area (R) of processor memory. The processor uses this information to run the instruction. See "Using the Control Structure" below.
- **Length** is the number of elements in the data block on which the file instruction operates. Enter any decimal number 1–1000.
- **Position** is the current location in the data block that the processor is accessing. You generally enter a zero to start at the beginning of a file.
- **Mode** is the number of file elements operated on per program scan. The mode lets you distribute file operation over multiple program scans. Specify one of the following:
  - for All mode, type an A
  - for Numerical mode, type a decimal number (1–1000)
  - for Incremental mode, type an I

  For more information about the different modes, see "Choosing Modes of File Operation" below.
- **Destination** is the address where the processor stores the result of the operation. The instruction converts to the data type specified by the destination address.
- **Expression** contains addresses, program constants, and operators that specify the source of data and the operations to be performed

When you enter destination and expression addresses, use the index prefix (#) for file addresses. The processor assigns and uses the offset value in module status (S:24) to process the file address. Omit the # prefix in a word or element address.

Important: Make sure the index value (positive or negative) does not cause the indexed address to exceed the file type boundry. The processor does not check this unless you use an indexed indirect address or exceed the data table area of memory. If the indexed address exceeds the data table area, the processor initiates a run-time error and sets a major fault. The processor does not check to see whether the indexed address crosses file types, such as N7 to F8.



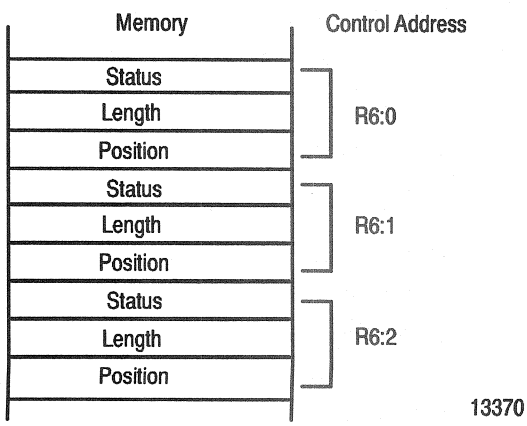
WARNING: File instructions manipulate the offset value stored at S:24. Make sure that you monitor or load the offset value you want prior to using an indexed address. Otherwise unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

For more information on indexed addressing, see the chapter on addressing data table files in the "Configuration and Maintenance" module.

Using the Control Structure

The control structure (data section R) controls the operation of the file instruction. Similar to a counter, it controls the file by length, position and status and control bits (Figure 8.1). You enter the control structure address (for example R6:0) at the Control entry when you program a FAL or FSC file instruction.

Figure 8.1
Example Control File R6:0





WARNING: Do not use the same control address for more than one instruction. Duplication of a control address could result in unpredictable operation, possibly causing damage to equipment and/or injury to personnel.

The control structure stores the following information:

- Status bits
- Length (.LEN) of the file (1–1000 words)
- Position (.POS) of the words that the processor is operating on

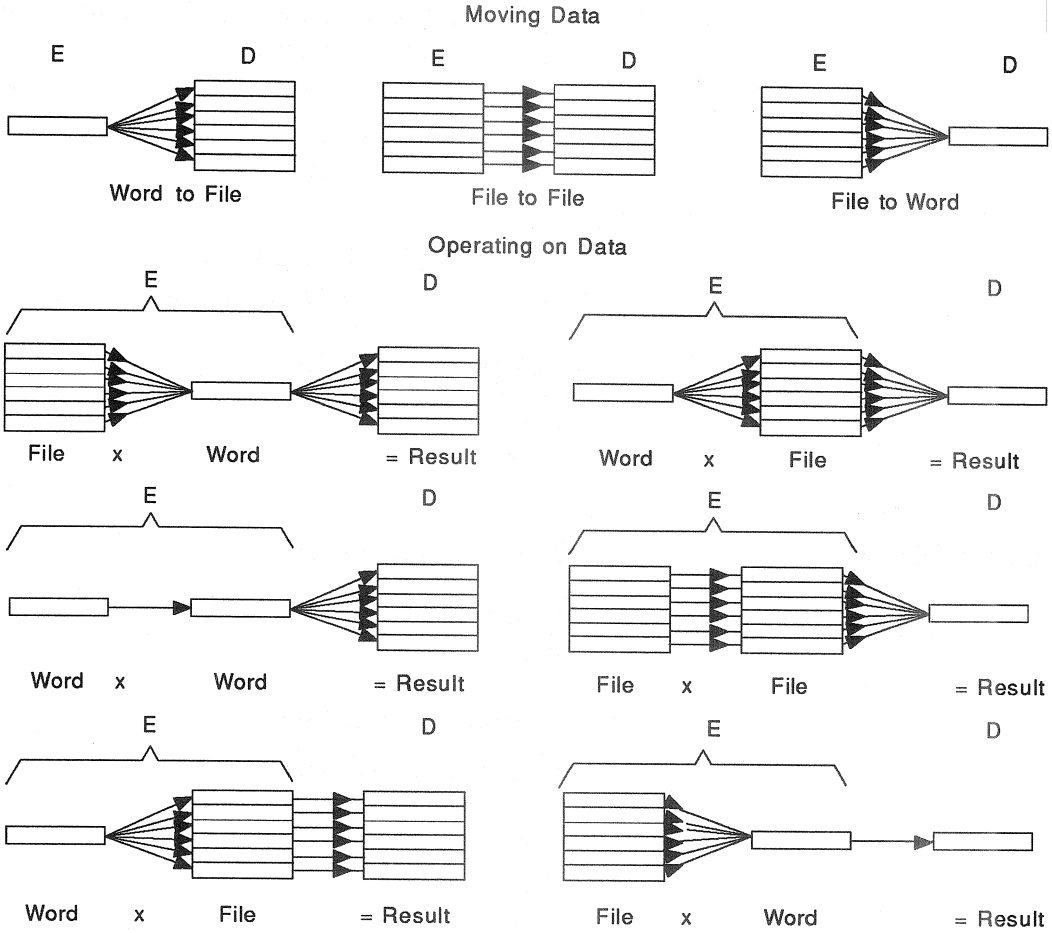
The FAL instruction and FSC instruction each has its own set of status bits. See the next chapter for the FAL or FSC instruction for a description of these status bits.

Manipulating File Data

Typical data manipulations with file instructions include:

- Copying data from a
  - source word to a destination file
  - source file to a destination file
  - source file to a destination word
- Operating on data from multiple sources such as
  - source words
  - source files
- Storing the result in a
  - destination file
  - destination word

The following example shows generic data manipulations used with file instructions (E = expression, D = destination, x = operation).

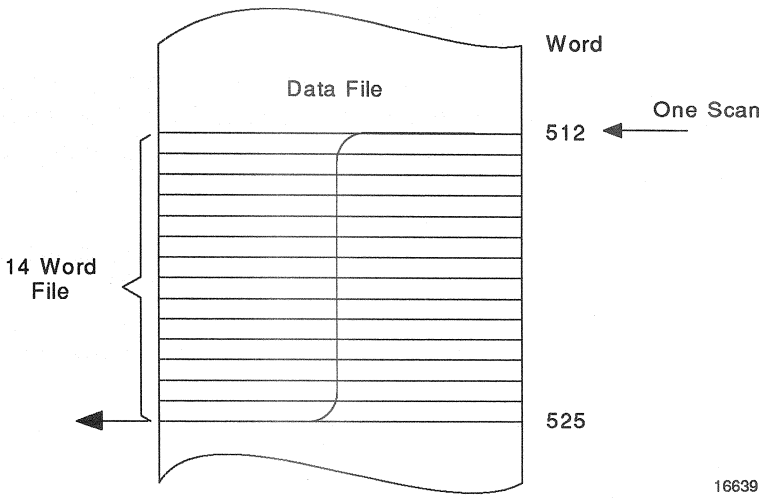


Choosing Modes of File Operation

The file mode you select when you program a FAL or FSC instruction tells the processor how to distribute the file operation over one or more program scans. Select one of the following modes:

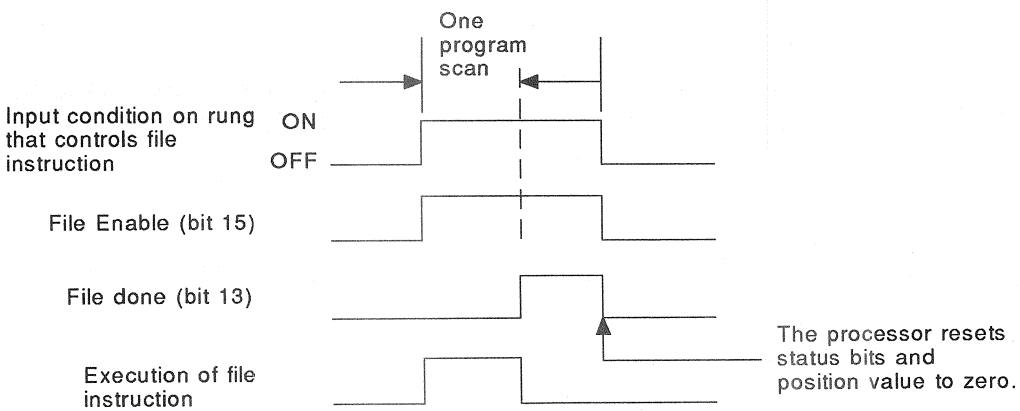
All Mode

Select all mode if you want to complete the file instruction operation in one program scan. Type an A for the mode parameter when you enter the instruction.



Operation begins when the rung goes from false to true. The position (.POS) value in the control element points to the element in the data block that the instruction is currently using. File operation stops when the function completes in one program scan or when the processor detects an error.

The following timing diagram shows the relationship between status bits and instruction operation. If the rung is false when the instruction completes, the done and enable bits are reset, and the position value is reset.

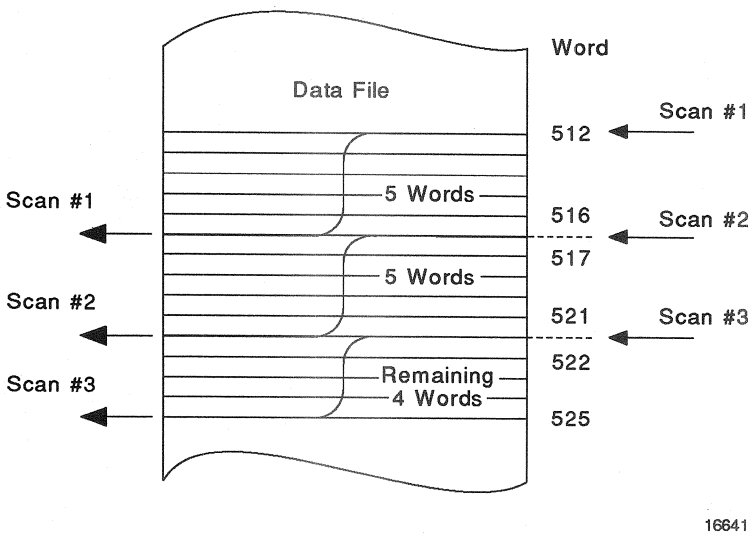


Numeric Mode

Select numerical mode if you want to distribute the file operation over a number of program scans. Enter the number of elements per scan (1–1000) for the mode parameter when you enter the file instruction. The number of elements that you enter must be less than or equal to the file length.

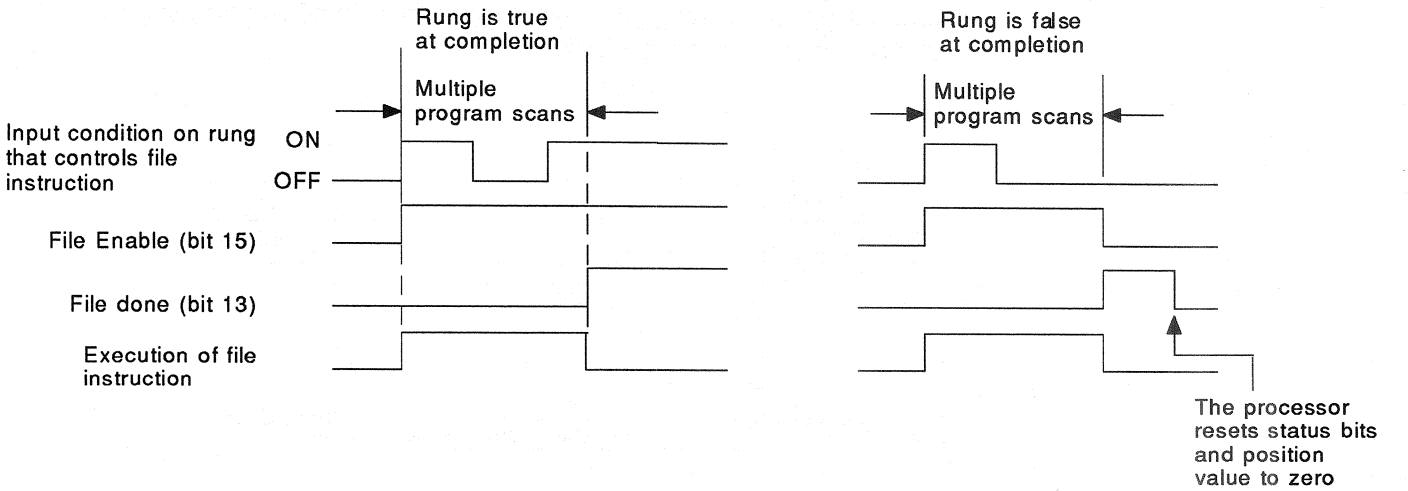
Operation begins when the rung goes from false to true. Once enabled, the instruction remains enabled for the number of program scans necessary to complete the file operation. Rung logic can change repeatedly without interrupting the file operation.

Each scan, the instruction operates on the number of elements equal to the rate value that you entered for the mode entry until the number of elements specified by the length value is scanned. In the last scan, the processor may operate on less than the number of elements you entered.



Important: Avoid using the results of a file instruction operating in numeric mode until the done bit is set because the data will be incomplete.

The following timing diagram shows the relationship between status bits and instruction operation.



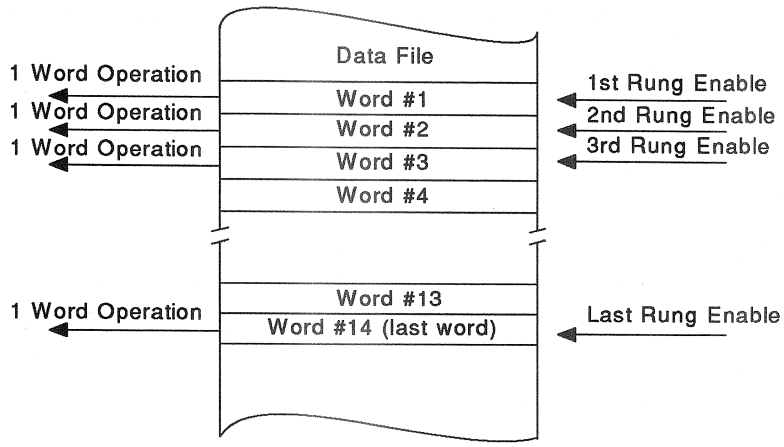
When the file operation is complete, the rung that contains the file instruction can be either true or false. If the rung is true at completion, the done and enable bits are both set. If the rung is false at completion, the done bit is set, the enable bit is reset during the scan after the last group of words is operated upon. The done bit and the position value are reset during the next program scan.

Special Case, Rate Per Scan = 1 : The difference between numerical mode with a rate per scan = 1 (words = 1) and incremental mode (described next) is:

- In numerical mode with any rate per scan, only one false–to–true rung transition is required for the instruction to operate on the entire file.
- Incremental mode requires a false–to–true rung transition for each word in the file.

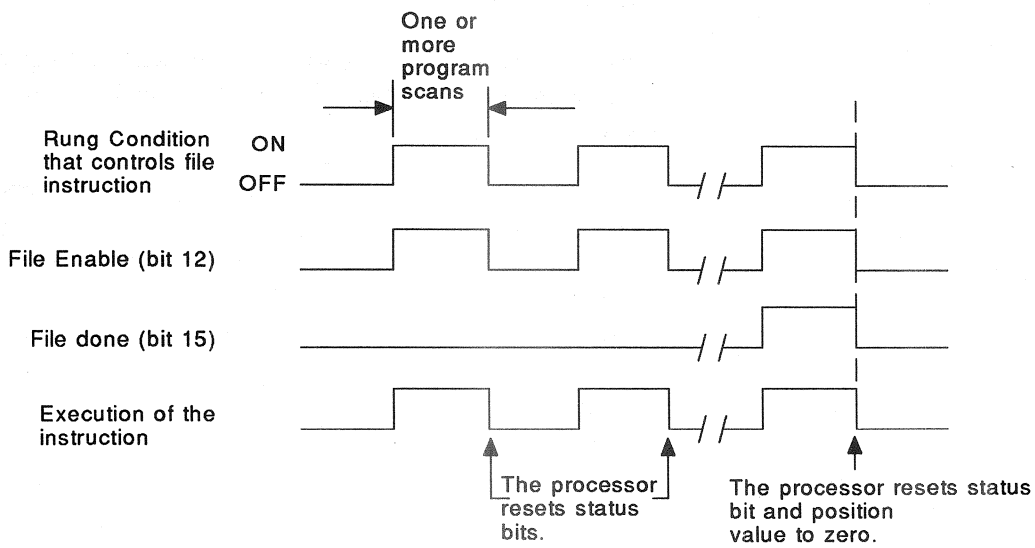
Incremental Mode

Select incremental mode if you want the file operation to manipulate one element each time the rung goes from false to true. Type an I for the mode parameter when you enter the instruction.



16643

The following timing diagram shows the relationship between status bits and instruction operation.



16644

Operation occurs only in the program scan when the rung goes from false to true. The operation only manipulates one file element. The enable bit is set when rung logic is true. The done bit is set the scan after the last element in the file has been operated upon. When the instruction has manipulated the last element and the rung goes false, the enable and done bits are reset and the control element's position (.POS) is cleared to zero. If the rung remains true for more than one program scan, file operation does not repeat.

Important: If you are operating on an entire file, avoid using the results of a file instruction using incremental mode until the done bit is set (the data will be incomplete).

Compare incremental and numerical modes. In both, the file operation is performed as the control element's position value sequentially points to each element in the file.

In numerical mode with a rate per scan of 1, the entire file operation is performed after one false-to-true transition in rung logic. The instruction operates on the entire file over the number of program scans equal to the file length plus 1. The instruction resets the done bit the scan after it operates on the last word in the file.

In incremental mode, rung logic must undergo a number of false-to-true transitions, one transition per element in the file.

Chapter 9

File Instructions FAL, FSC, COP, FLL

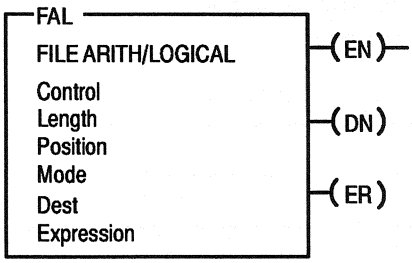
Using File Instructions

The file instructions let you perform operations on file data and compare file data. Table 9.A lists the available file instructions.

Table 9.A
Available File Instructions

If You Want to:	Use This Operation:	Found on Page:
Perform arithmetic, logic, shift, and function operations on file data	FAL	9-1
Perform search and compare operations on file data	FSC	9-13
Copy the contents of a file into another file	COP	9-17
Fill a file with specific values	FLL	9-18

File Arithmetic and Logic (FAL)



The File Arithmetic and Logic (FAL) instruction performs copy, arithmetic, logic, and function operations on the data stored in files. The FAL instruction performs the same operations as the compute (CPT) instruction. The difference is that the FAL instruction performs operations on multiple words, while the CPT instruction handles single words.

Description:
The FAL instruction is an output instruction that performs the operations defined by source addresses and operators you write in the expression. The instruction writes the results into a destination address.

You select how the processor distributes the operation over one or more program scans by your selection of instruction mode. For more information about modes of file operation, see the previous chapter "File Instruction Concepts."

The FAL instruction automatically converts the data type at the source addresses to the data type that you specify in the destination address.

You can use this instruction to perform operations such as:

- Zero a file
- Copy data from one file to another
- Make arithmetic or logic computations on data stored in files
- Unload a file of error codes one at a time for display

WARNING: File instructions manipulate the offset value stored at S:24. Make sure that you monitor or load the offset value you want prior to using an indexed address. Otherwise unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

If you have not already done so, review the basic concepts of file operation in the previous chapter "File Instruction Concepts." For more information on using indexed addresses, see the chapter on addressing data table files in the "Configuration and Maintenance" module.

Table 9.B
FAL Operations

Type	Operator	Description	Example Operation
Copy	none	copy from A to B	enter source address in the expression enter destination address in destination
Clear	none	set a value to zero	0 (enter 0 for the expression)
Arithmetic	+	add	2 + 3 2 + 3 + 7 PLC-5/30, -5/40, -5/60
	-	subtract	12 - 5 (12 - 5) - 1 PLC-5/30, -5/40, -5/60
	*	multiply	5 * 2 6 * (5 * 2) PLC-5/30, -5/40, -5/60
	|	divide	24 | 6 (24 | 6) * 2 PLC-5/30, -5/40, -5/60
	-	negate	- N7:0
	SQR	square root	SQR N7:0
	**	exponential (x to the power of y)	10**3 (PLC-5/30, -5/40, -5/60 only)
Bitwise	AND	bitwise AND	D9:3 AND D10:4
	OR	bitwise OR	D9:4 OR D9:5
	XOR	bitwise exclusive OR	D10:10 XOR D10:11
	NOT	bitwise complement	NOT D9:4
Conversion	FRD	convert from BCD to binary	FRD N7:0
	TOD	convert from binary to BCD	TOD N7:0

Using Status Bits

To use the FAL instruction correctly, the ladder program must examine and control status bits in the control element. You address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	by a false-to-true rung transition and indicates the instruction is enabled. In incremental mode, the .EN bit follows the rung condition. In numerical and ALL modes, the .EN bit remains set until the instruction completes its operation, regardless of the rung condition. The .EN bit is reset when the rung goes false and the instruction completes its operation.
Done .DN (bit 13)	after the instruction has operated on the last set of words. In numerical mode if the instruction is false at completion, it resets the .DN bit one program scan after the operation is complete. Otherwise, the .EN bit resets the .DN bit.
Error .ER (bit 11)	when the operation generates an overflow. The instruction stops until the ladder program resets the .ER bit. When the processor detects an error, the position value stores the number of the word that faulted.

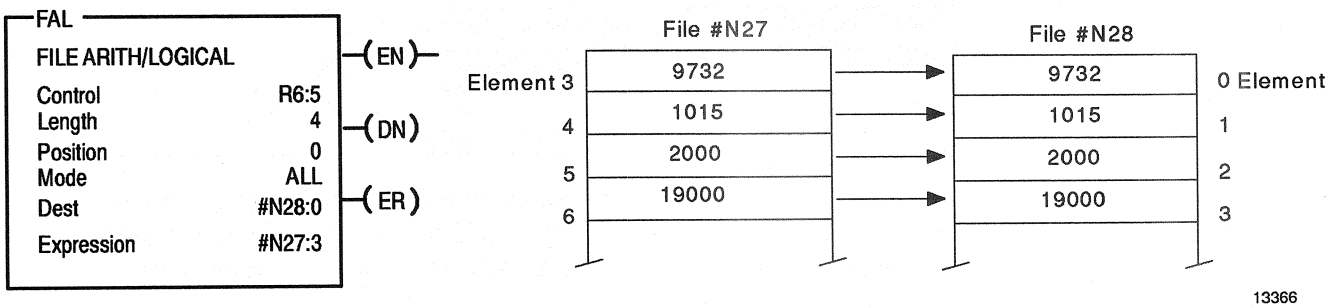
FAL Copy Operations

The FAL copy operation copies data:

- Between files
- From a word to a file
- From a file to a word

To copy data with the FAL copy operation, enter the source address or program constant in the expression and the destination address in the destination.

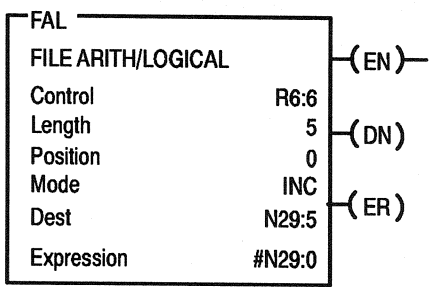
File-to-File Copy Example:

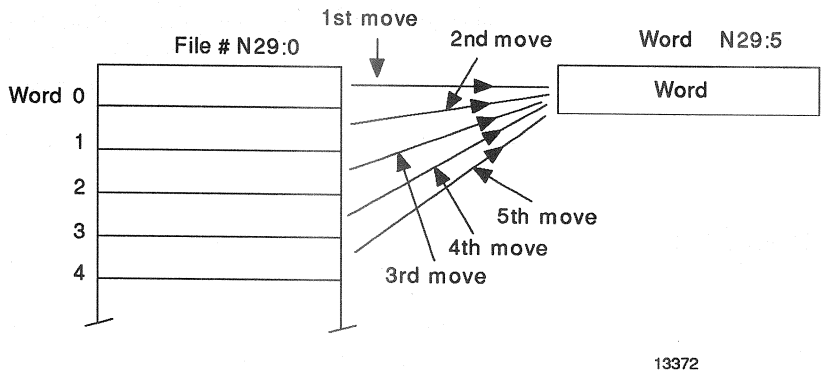


This Parameter:	Tells the Processor:
Control (R6:5)	What control structure controls the operation
Length (4)	To move four words
Position (0)	To start at the source address
Mode (ALL)	To execute the length in one program scan
Destination (#N28:0)	Where to write the data (the # indicates that the operation is to be performed on a file)
Expression (#N27:3)	Where to read the data (the # indicates that the operation is to be performed on a file)

When the rung goes true, the processor reads four elements of integer file N27 word by word starting at element 3, and writes the image to integer file N28 starting at element 0. It writes over any data in the destination file.

File-to-Word Copy Example:





This Parameter:	Tells the Processor:
Control (R6:6)	What control structure controls the operation
Length (5)	To copy five words
Position (0)	To start at the source address
Mode (incremental)	To copy one word each time the rung goes true
Destination (N29:5)	Where to write the data (word address)
Expression (#N29:0)	Where to read the data (the # indicates that the operation is to be performed on a file)

With each false-to-true rung transition, the processor reads one element of integer file N29 starting at element 0, and writes the image into element 5 of integer file N29. The instruction writes over any data in the destination.

A word-to-file move is similar except that the instruction copies data from a word address into a file. The word address can be in the same or a different file.

FAL Arithmetic Operations

Some arithmetic operations require 2 or 3 addresses: 2 for the expression and 1 for the result or destination (addresses can represent files or elements). The specific operation is determined by the arithmetic symbol you enter in the expression. You can perform multiple arithmetic operations on file data (integer or floating point) with the following operators:

Operator:	Meaning:	Operator:	Meaning:
+	add	/	divide
-	subtract	-	negate
*	multiply	0	clear

To perform multiple arithmetic operations in a single FAL instruction, you enter the operators, source addresses, or program constants in the expression, and the result address in the destination. The processor executes math operators in a predetermined order. For more information about order of operation, see "Order of Operation" in the chapter "Compute Instructions."

Upper and Lower Limits

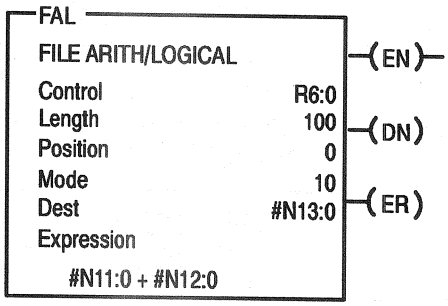
The limits of data being mathematically manipulated depend on the type of file in which the data is stored. The following guidelines apply:

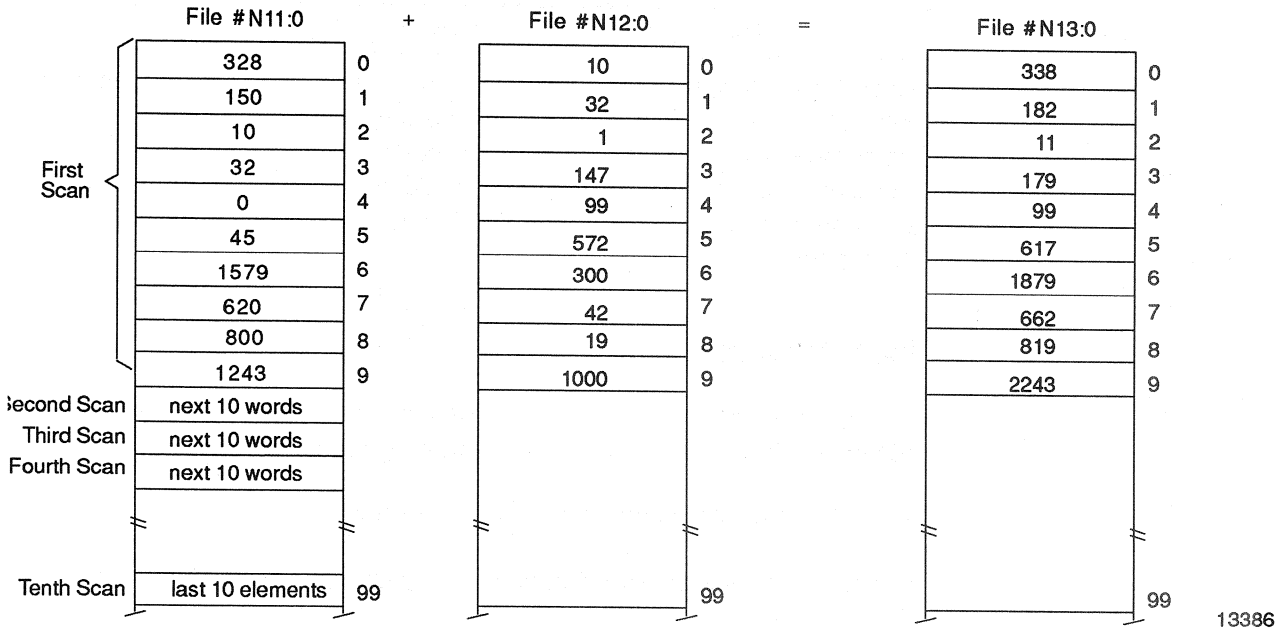
- all data except floating point is signed integer
- negative values are stored in two's complement
- floating point numbers are formatted as a subset of IEEE single-precision floating point

Type of File:	Range Stored in Word:
bit	-32768 to +32767 for integers
integer	-32, 768 to +32, 767
timer	0 to +32, 767
counter	-32, 768 to +32, 767
control	0 to +32, 767
floating point	$\pm 1.1754944e^{-38}$ to $\pm 3.4028237e^{+38}$

The processor declares an error when it detects that the result of an operation exceeds either the lower or upper limit of the destination word in which it is stored. The overflow bit is set in the processor's status file (S:0/1). The instruction also sets the error bit in the status byte of its control word.

Addition Example:

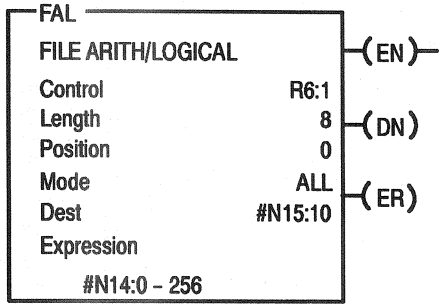


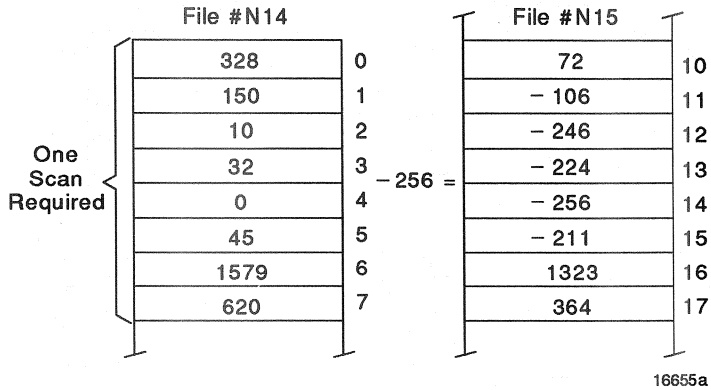


This Parameter:	Tells the Processor:
Control (R6:0)	What control structure controls operation
Length (100)	To operate on one hundred elements
Position (0)	To start at the source address
Mode (10)	To execute the data in 10 words per scan
Destination (#N13:0)	Where to write the result data
Expression (#N11:0 + #N12:0)	The operators, program constants, and source addresses

When the rung goes true, the processor adds 100 values in file #N11:0 to the corresponding values in file #N12:0, using the numerical mode of 10 words per scan. The operation is performed in 10 scans and the instruction sequentially adds the values in the expression, storing the result in file #N13:0.

Subtraction Example:

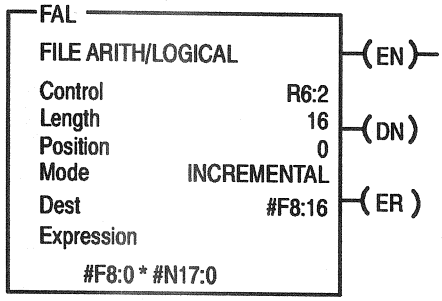


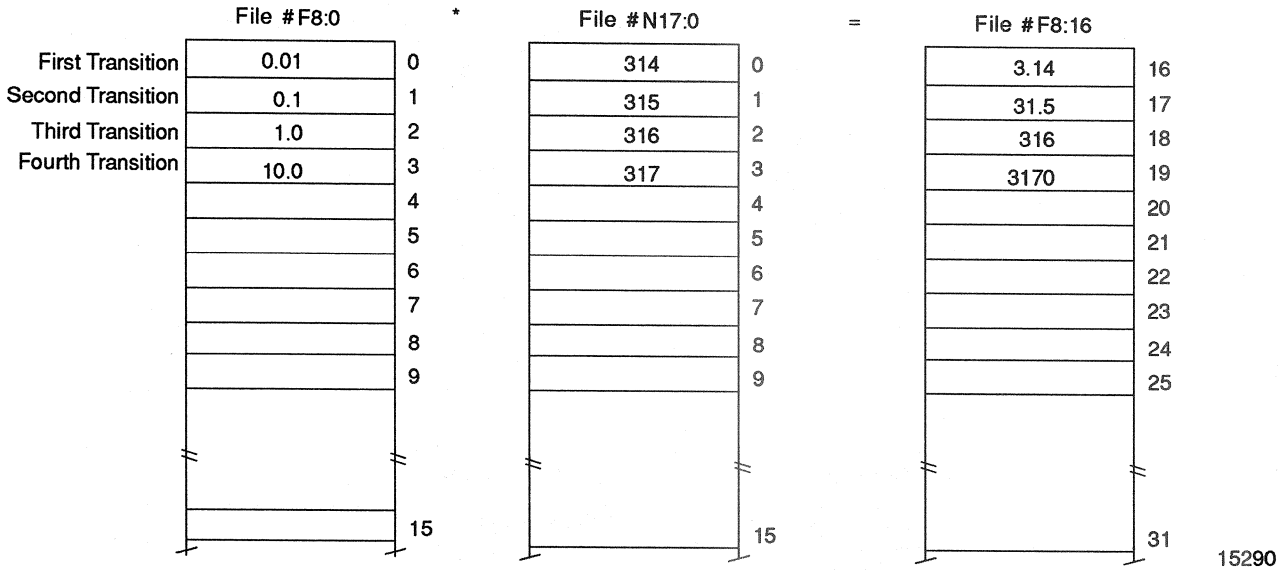


This Parameter:	Tells the Processor:
Control (R6:1)	What control structure controls operation
Length (8)	To operate on eight words
Position (0)	To start at the source address
Mode (ALL)	To execute the data in one program scan
Destination (#N15:10)	Where to write the result data
Expression (#N14:0 - 256)	The operators, program constants, and source addresses

When the rung goes true, the processor reads eight elements of integer file N14 word by word starting at element 0, subtracts a program constant (256) from each, and writes the result into destination file N15 starting at element 10, all in one scan.

Multiplication Example:

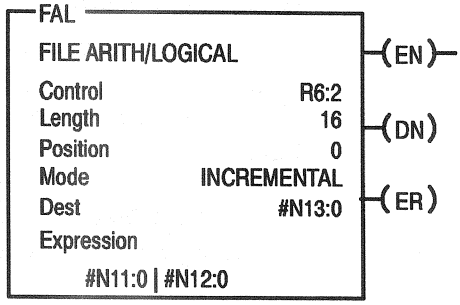


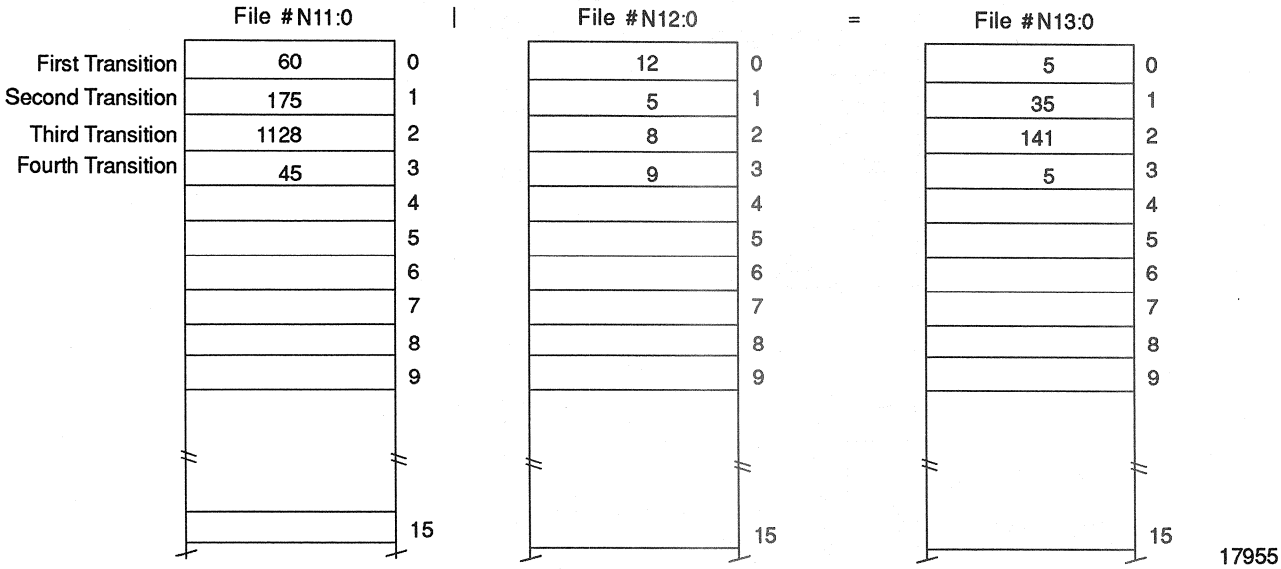


This Parameter:	Tells the Processor:
Control (R6:2)	What control structure controls operation
Length (16)	To operate on sixteen words
Position (0)	To start at the source address
Mode (INCREMENTAL)	To execute using incremental mode
Destination (#F8:16)	Where to write the result data
Expression (#F8:0 * #N17:0)	The operators, program constants, and source addresses

When the rung goes true, the processor multiplies 16 values in file #F8:0 by the corresponding values in file #N17:0, using incremental mode. One multiplication is performed for each false to true transition. The operation requires 16 transitions, storing the result in file #F8:16.

Division Example:



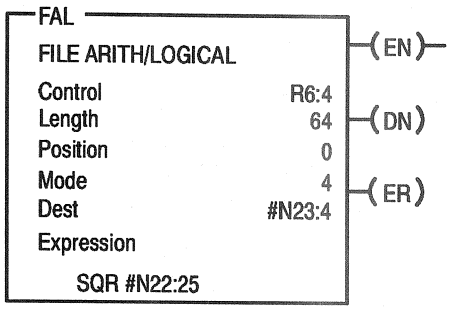


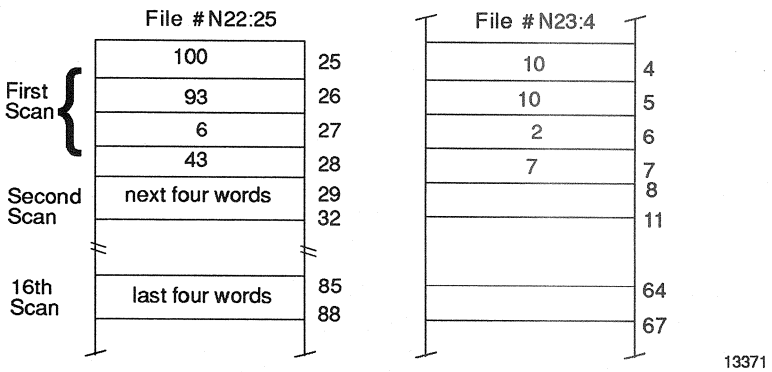
This Parameter:	Tells the Processor:
Control (R6:2)	What control structure controls operation
Length (16)	To operate on sixteen words
Position (0)	To start at the source address
Mode (INCREMENTAL)	To execute using incremental mode
Destination (#N13:0)	Where to write the result data
Expression (#N11:0 | #N12:0)	The operators, program constants, and source addresses

When the rung goes true, the processor divides 16 values in file #N11:0 by the corresponding values in file #N12:0, using incremental mode. One division is performed for each false to true transition. The operation requires 16 transitions, storing the result in file #N13:0.

File Square Root Example:
When rung conditions go from false to true, the instruction obtains the positive square root of the value in the expression. The rate is determined by the file mode you select. The result of each square root operation is stored in the corresponding element in the destination, one element at a time.

Note that the processor takes the square root of the absolute value (if the sign is negative, the processor disregards the sign).

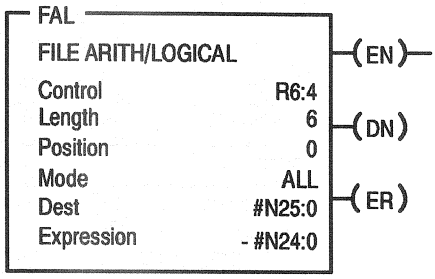




On the first rung transition, the square root is obtained for the first four elements in the expression. The instruction takes the square root of the next four elements in each subsequent scan until the operation is complete.

This Parameter:	Tells the Processor:
Control (R6:4)	What control structure controls the operation
Length (64)	To take the square root of 64 words
Position (0)	To start at the source address
Mode (4)	To move 4 words each time the rung goes true
Destination (#N23:4)	Where to write the result data
Expression (SQR #N22:25)	The square root of #N22:25

Negate Example:



When rung conditions go from false to true, the instruction changes the sign (+ or –) of each value and moves each value in the expression into the corresponding location in the destination file. The rate is determined by the mode of operation you select.

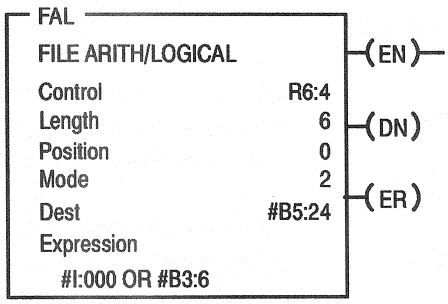
FAL Logic Operations

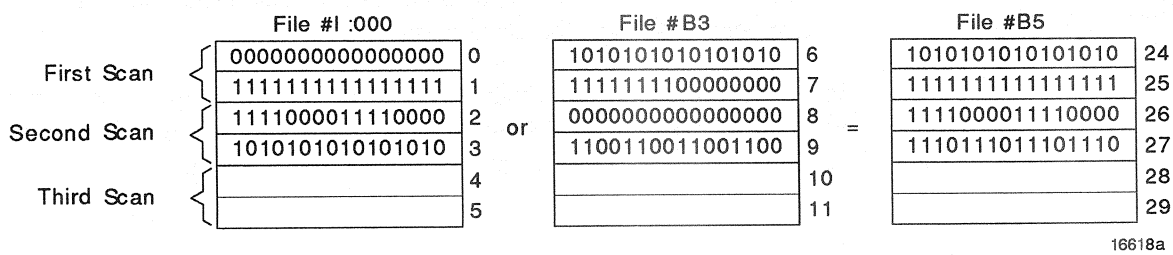
You can perform multiple logic operations on binary file data with the following bitwise logic operators:

- AND
- OR
- XOR
- NOT

To perform multiple logic operations, you enter the operators, source addresses, or program constants in the expression, and the result address in the destination.

Logical OR Example:





This Parameter:	Tells the Processor:
Control (R6:4)	What control structure controls the operation
Length (6)	To OR 6 words
Position (0)	To start at the source address
Mode (2)	To move 2 words each scan
Destination (#B5:24)	Where to write the result data
Expression (#I:000 OR #B3:6)	The operator(s) and source addresses

After the rung goes true, on each scan the processor reads two elements from input image file I starting at element 0, ORs them with the corresponding two elements in binary file B3 starting at element 6, and writes the result into destination file B5 starting at element 24. This instruction only operates the first 3 scans (length of 6 / mode of 2) after the rung goes true.

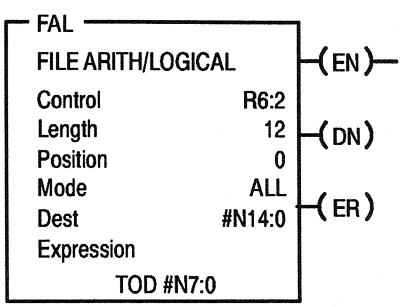
The processor executes logic operators in a predetermined order. For more information about order of operations, see "Order of Operation" in the chapter "Compute Instructions."

FAL Functions

The FAL instruction can perform other functions. These functions are:

- Convert from integer to BCD (TOD)
- Convert from BCD to integer (FRD)
- Clear a file (replace each value with a zero)

Example: File Convert to BCD

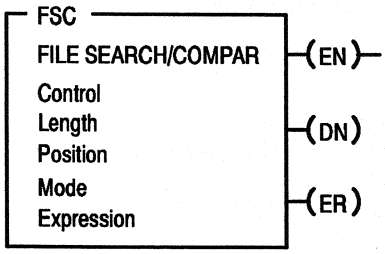


File Convert to BCD (TOD):
When rung conditions go from false to true, the processor converts the value in the expression form integer to BCD. The rate is determined by the file mode that you select. The result of the operation is stored in the corresponding element in the destination.

File Convert from BCD (FRD):
When rung conditions go from false to true, the processor converts the value in the expression from BCD to integer. The rate is determined by the file mode that you select. The result of the operation is stored in the corresponding word in the destination.

Important: Convert BCD values to integer before manipulating them in your ladder program; if you do not convert the values, the processor manipulates them as integer and their BCD value is lost.

File Search and Compare (FSC)



The File Search and Compare (FSC) instruction performs search and compare operations. The FSC instruction performs the same operations as the compare (CMP) instruction, including complex expressions. The difference is that the FSC instruction performs logical operations on files, while the CMP instruction operates on words or elements. Also, the FSC instruction is an output instruction, while the CMP instruction is an input instruction.

Description:
The FSC instruction is an output instruction that compares values in source files, element by element, for the logical operations you specify in the expression. When the instruction finds the specified comparison is true, it sets the found bit .FD, and records the position .POS where the true comparison was found. The inhibit bit .IN is set to prevent any further searching of the files.

Your ladder program must examine the found bit .FD and the position .POS to take appropriate action. The ladder program must reset the inhibit bit .IN, so the instruction can continue.

You select how the processor distributes the operation over one or more program scans by your selection of instruction mode. For more information about modes of file operation, see the "File Instruction Concepts" chapter in this module.

You can use this instruction to performs operations such as:

- Set high and low process alarms for multiple analog inputs
- Compare batch variables against a reference file before starting a batch operation

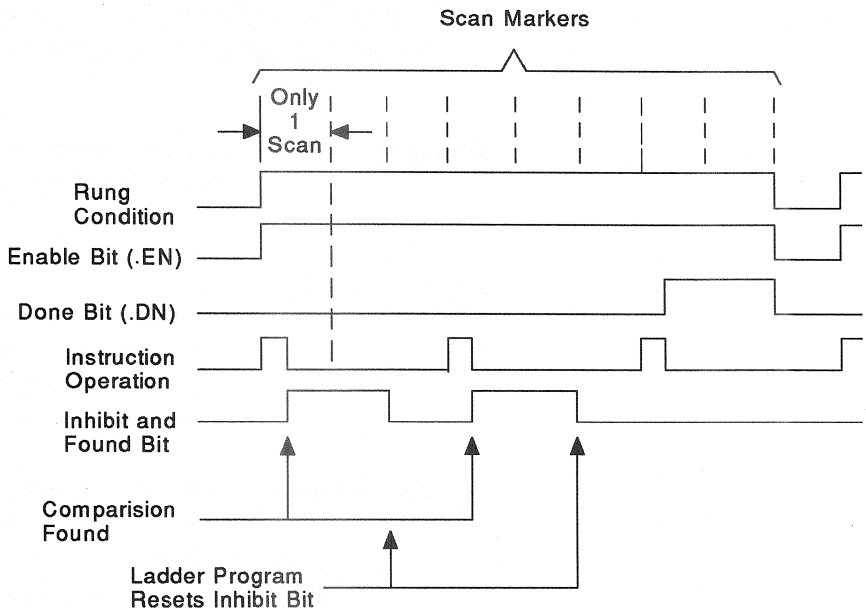
If you have not already done so, review the basic concepts of file operation in "File Concepts for FAL and FSC Instructions."

Using Status Bits

To use the FSC instruction correctly, your ladder program must examine and control status bits in the control element. You address these bits by mnemonic.

This bit:	Is Set:
Enable .EN (bit 15)	by a false-to-true rung transition and indicates the instruction is enabled. In incremental mode this bit follows the rung condition. In numerical and ALL modes, this bit remains set until the instructions completes its operation, regardless of the rung condition. The .EN bit is reset when rung conditions go false, but only after the instruction has set the done bit.
Done .DN (bit 13)	after the instruction has operated on the last set of words In numerical mode if the instruction is false at completion, it resets the .DN bit one program scan after the operation is complete. Otherwise, the .EN bit resets the .DN bit.
Error .ER (bit 11)	when the operation generates an overflow. The instruction stops until the ladder program resets this bit When the processor detects an error, the position value stores the number of the element that faulted.
Inhibit .IN (bit 9)	when the processor detects a true comparison Your ladder program must reset this bit to continue the search after taking an action initiated by examining the .FD bit. The ladder program must reset this bit to continue operation.
Found .FD (bit 8)	when the processor detects a true comparison. The processor stops the search and also sets the inhibit bit .IN. The .FD bit is the output of the FSC instruction.

The following timing diagram for ALL mode shows relationships between status bits and instruction execution when the instruction finds two true conditions.



For more information about how the FSC instruction responds when it finds no true comparisons, see the timing diagrams in "Choosing Modes of File Operation" in the previous chapter.

FSC Search and Compare Operations

The FSC instruction performs these comparisons on file data according to how you specify them in the Expression. (Complex expressions are valid in PLC-5/30, -5/40 and -5/60 processors.)

Comparison:	Example Expression:
Search Equal	#N50:0 = #N51:0
Search Not Equal	#N52:0 <> N52:11
Search Less Than	#B3:100 < #N53:0
Search Less Than or Equal	#F60:0 <= F60:12
Search Greater Than	#N54:0 > 256
Search Greater Than or Equal	F61:10 >= #N61:0

Data Conversion

The processor compares files of different data types by internally converting data into its binary equivalent before performing the comparison. The processor treats the following data types as integer: timer, status, bit, counter, input, ASCII, control, output, BCD.

Important: When you compare floating point and integer values in the FSC instruction, limit the comparisons to "less than or equal" and "greater than or equal".

Important: Use ASCII and BCD for display only, and not as values. Since the processor interprets them as integer, they may lose their meaning if you enter them as values.

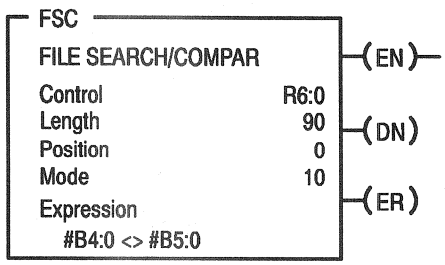
For the order in which the instruction performs logical operations, see the section "Order of Operation" in the chapter "Compute Instructions" in this module.

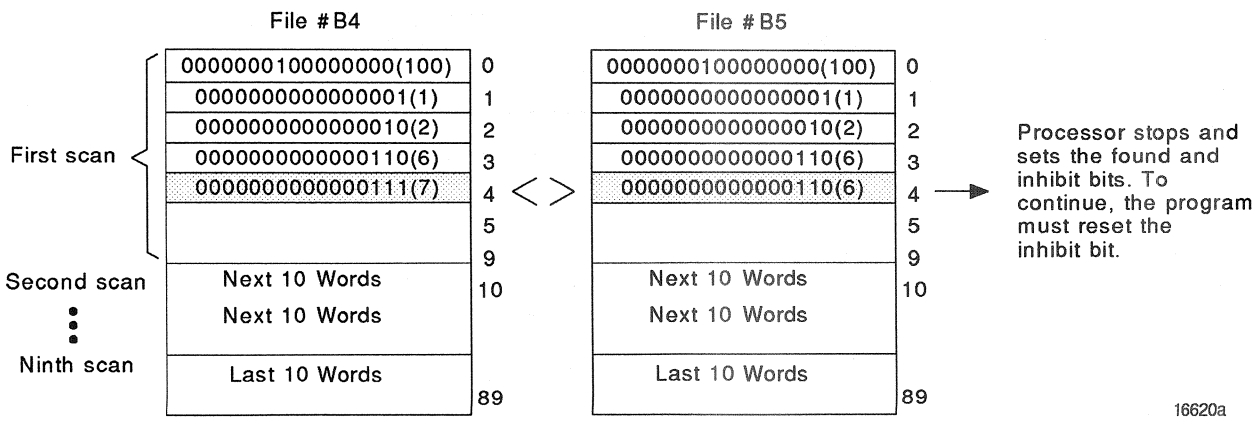
File Search Operation

When the rung condition goes from false to true, the desired comparison is performed on data addressed in the expression. Elements are compared in ascending order, starting at the beginning. The rate is determined by the mode of operation that you specify.

The done bit (bit 13) is set after the instruction has compared the last pair. If the instruction is true at completion, the done bit is reset when the rung goes false. In numerical mode, however, if the instruction is false at completion, the done bit is set one program scan after the operation is complete.

Example of Search Not Equal:





This Parameter:	Tells the Processor:
Control (R6:0)	What control structure controls the operation
Length (90)	To search through 90 words
Position (0)	To start at source addresses
Mode (10)	To search 10 words per program scan
Expression (#B4:0 <> #B5:0)	The comparison to perform and the source addresses

When a rung containing the FSC instruction goes from false to true, the processor performs the not-equal-to comparison between words of the source files B4 and B5, starting with word 0 for each file. The number of words compared per program scan (10 in this example) is determined by the mode you select.

When the processor finds that corresponding source elements are not equal (element B4:4 and B5:4 in this example), the processor stops the search and sets the found .FD and inhibit .IN bits so your ladder program can take appropriate action. To continue the search comparison, you must reset the .IN bit.

File Copy (COP)

COP
COPY FILE
Source
Dest
Length

Description:
The COP instruction is an output instruction that copies the values in the source file into the destination file. The source remains unchanged. The COP instructions does not use status bits. If you need an enable bit, program a parallel output that uses a storage address.

You can perform file shifts by specifying a source address one or more greater than the destination address within the same file. This shifts data to lower addresses. You can use this technique to store data of assembly line events instead of using FIFO load/unload instructions.

The COP instruction does not write over file boundaries. Any overflow data is lost. Also, no data conversion occurs if the source and destination files are different types; use the same file type for each.

The destination file type determines the number of words per element that the instruction transfers. For example, if the destination file type is counter and the source file type is integer, three integer words are transferred for each element in the counter-type file.

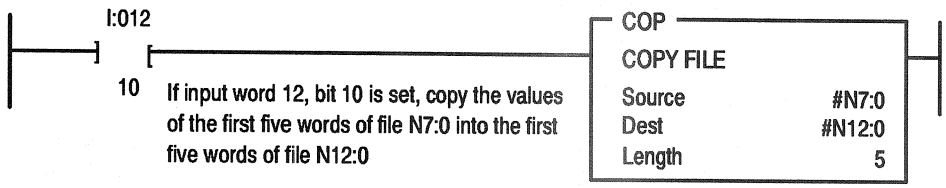
Fast operation occurs in microseconds:
- integer – 5 microseconds
- floating-point – 8 microseconds
- timer, counter, control – 11 microseconds

Entering Parameters

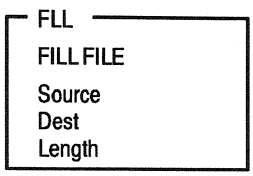
To program the COP instruction, you must provide the processor with the following:

Parameter:	Definition:
source	the address of the source file. The source remains unchanged.
destination	address of the destination file. The instruction writes over any data already stored at the destination.
length	the number of the words/elements to overwrite in the destination file.

COP Example:



File Fill (FLL)



Description:
The FLL instruction is an output instruction that fills the words of a file with a source value. The source remains unchanged. The FLL instructions does not use status bits. If you need an enable bit, program a parallel output that uses a storage address.

The FLL instruction does not write over file boundaries. Any overflow data is lost. Also, no data conversion occurs if the source and destination files are different types; use the same file type for each.

The destination file type determines the number of words per element that the instruction transfers. For example, if the destination file type is counter and the source file type is integer, three integer words are transferred for each element in the counter-type file.

Fast operation occurs in microseconds:

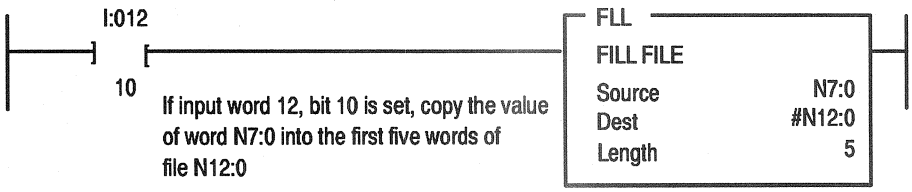
- integer – 5 microseconds
- floating-point – 8 microseconds
- timer, counter, control – 11 microseconds

Entering Parameters

To program the FLL instruction, you must provide the processor with the following:

Parameter:	Definition:
source	the address of the source word or a program constant. The source remains unchanged.
destination	the address of the destination file. The instruction writes over any data already stored at the destination.
length	the number of the words/elements to fill in the destination file

FLL Example:



Words are copied from the specified source file into the specified destination file every scan that the rung is true. They are copied (in ascending order with no transformation of data) up to the specified number or until the last element of the destination file is reached, whichever occurs first.

Be sure that you accurately specify the starting address and length of the data block you are copying. The instruction will not write over a file boundry (such as between files N16 and N17) at the **destination**. The overflow would be lost.

Chapter 10

Diagnostic Instructions FBC, DDT, DTR

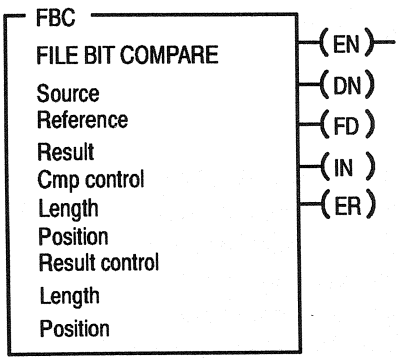
Using Diagnostic Instructions

The diagnostic instructions let you detect problems with data in your programs. Table 10.A lists the available diagnostic instructions.

Table 10.A
Available Diagnostic Instructions

If You Want to:	Use This Operation:	Found on Page:
Compare I/O data against a known, good reference and record any mismatches	FBC	10-1
Compare I/O data against a known, good reference, record any mismatches, and update the reference file to match the source file	DDT	10-1
Pass source data through a mask and compare the result to reference data, and then write the source word into the reference address of the next comparison.	DTR	10-6

File Bit Comparison (FBC) and Diagnostic Detect (DDT)



The FBC and DDT diagnostic instructions are output instructions that you use to monitor machine or process operations to detect malfunctions.

Table 10.B
Available Diagnostic Instructions

If You Want to Detect Malfunctions By:	Use This Instruction:
Comparing bits in a file of real-time inputs with a reference bit file that represents correct operation	FBC
Change-of-state diagnostics	DDT

Description:
Both the FBC and DDT instructions compare bits in a file of real-time machine or process values (input file) with bits in a reference file, detect deviations, and record mismatched bit numbers. These instructions record the position of each mismatch found and place this information in the result file.

The difference between the DDT and FBC instruction is that each time the DDT instruction finds a mismatch, the processor changes the reference bit to match the source bit. You can use the DDT instruction to update your reference file to reflect changing machine or process conditions.

Selecting the Search Mode

You can select whether the diagnostic instruction searches for one mismatch at a time or whether the it searches for all mismatches during one program scan.

One Mismatch at a Time
Operation of either instruction follows the rung condition. With each false-to-true rung transition, the instruction searches for the next mismatch between the input and reference files. Upon finding a mismatch, the instruction stops and sets the found bit .FD. Then the instruction enters the position number of the mismatch into the result file.

The DDT instruction also changes the status of the reference bit to match the status of the corresponding input bit. The instruction resets the found bit when the rung goes false.

When the instruction reaches the end of the file, the done bit (bit 13 DN of the compare control element) is set. Then, when the rung goes false, the instruction resets:

- enable bit
- found bit (if set)
- compare done bit
- result done bit (if set)
- both control counters

To enable this mode of operation, set the inhibit bit (.IN = 1) either by ladder program or manually before program execution.

All per Scan
Operation of either instruction starts with a false-to-true rung transition. The instruction searches for all mismatches between the input and reference files in one program scan. Upon finding mismatches, the instruction enters the position numbers of mismatched bits into the result file in the order it finds them. After reaching the end of the input and reference files, the instruction sets the .FD bit if it finds at least one mismatch. Then the instruction sets the .DN bit.

If you select a length for the result file that cannot hold all detected mismatches (if the result file fills), the instruction stops and requires another false-to-true rung transition to continue operation. The instruction wraps the new mismatched bit positions into the beginning of the result file writing over the old.

After completing the comparison and when the rung goes false, the instruction resets:

- enable bit
- found bit (if set)
- compare done bit
- result done bit (if set)
- both control counters

To enable this mode of operation, reset the inhibit bit (.IN = 0) by ladder program or manually before program execution.

Entering Parameters

To program these instructions, you need to provide the processor with the following information:

- **Source** is the indexed address of your input file.
- **Reference** is the indexed address of the file that contains the data with which you compare your input file.
- **Result** is the indexed address of the file where the instruction stores the position (bit) number of each detected mismatch.
- **Cmp Control** is the address of the comparison control structure in the control area (R) of memory that stores status bits, the length of the source and reference files (both should be the same), and the current position during operation.

  Use the compare control address with mnemonic when you address these parameters:

  - **Length (.LEN)** is the decimal number of bits to be compared in the source and reference files. Remember that bits in I/O files are numbered in octal 00-17, but that bits in all other files are numbered in decimal 0-15.
  - **Position (.POS)** is the current position of the bit to which the instruction points. Enter a value only if you want the instruction to start at an offset concurrent with a control file offset for one scan.

- **Result Control** is the address of the result control structure in the control area (R) of memory for the result file. This file stores the bit position number each time the instruction finds a mismatch between source and reference files.

  Use the result control address with mnemonic when you address these parameters:

  - **Length (.LEN)**is the decimal number of elements in the result file. Make the length long enough to record the maximum number of expected mismatches.

- **Position (.POS)** is the current position in the result file. Enter a value only if you want the instruction to start at an offset concurrent with a control file offset for one scan.

WARNING: Do not use the same address for more than one control structure. Duplication of these addresses could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

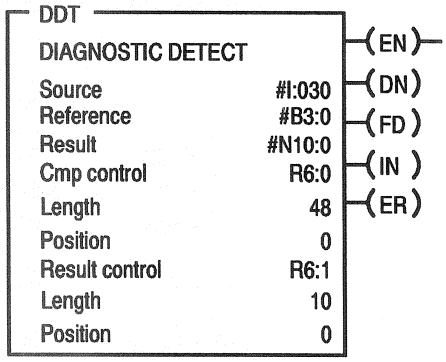
Using Status Bits

To use the FBC or DDT instruction correctly, your ladder program must examine and control bits in both the comparison and result control elements. You address these bits by mnemonic.

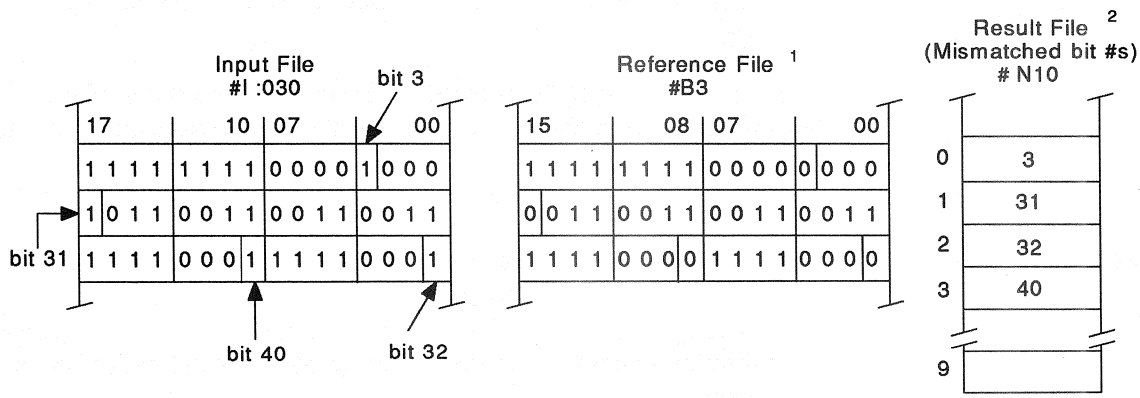
Bit:	Function:
Comparison Control Bits	
Enable .EN (bit 15)	starts operation on a false-to-true rung transition If the inhibit bit .IN is set for one-at-a-time operation, the ladder program must toggle the .EN bit after the instruction detects each mismatch.
Done .DN (bit 13)	is set when the processor reaches the end of the source and reference files
Error .ER (bit 11)	is set when the processor detects an error and stops operation of the instruction. For example, an error occurs if the length (.LEN) is less than or equal to zero or if the position (.POS) is less than zero. The ladder program must reset the .ER bit if the instruction detects an error.
Inhibit .IN (bit 09)	determines the mode of operation When this bit is reset, the processor detects all mismatches in one scan. When this bit is set, the processor stops the search at each mismatch and waits for the ladder program to re-enable the instruction before continuing the search.
Found .FD (bit 08)	is set each time the processor records a mismatch bit number in the result file (one-at-a-time operation) or after recording all mismatches (all per scan).
Result Control Bit	
Done .DN (bit 13)	is set when the result file fills The instruction stops and requires another false-to-true rung transition to reset the result .DN bit and then continue. If the instruction finds another mismatch, it wraps the new position number around to the beginning of the file, writing over previous position numbers.

After the FBC or DDT instruction sets the compare .DN bit, the instruction is reset when the rung's input conditions go false. The instruction resets its status bits and both control elements.

Diagnostic Detect and File Bit Comparison Instruction Example:
(same for both)



For example, the DDT instruction above compares the bits in the source file (#I:030) with the bits in the reference file (#B3), recording the mistmatched bit positions in the result file (#N10):



The FBC and DDT instructions detect mismatches and record their locations by bit number in a result file.

[1] The DDT instruction changes the status of the corresponding bit in the reference file to match the input file when it detects a mismatch.

[2] The length of the result file is the length that you enter for RSLT CONTROL.

16657a

This Parameter:	Tells the Processor:
Source (#I:030)	Where to find input data for comparison
Reference (#B3:0)	Where to find the reference file
Result (#N10:0)	Where to store mismatched bit numbers
CMP Control (R6:0)	What control structure controls the comparison
Length (48)	The number of bits to be compared
Position (0)	To start at the beginning of the file
Result Control (R6:1)	What control structure controls the result
Length (10)	The number of words reserved for mismatches
Position (0)	To start at the beginning of the file

When a rung containing the FBC instruction goes from false to true, the processor compares the bits in the source and reference files. The number of bit mismatches detected per program scan is determined by the mode of operation you select with the .IN bit.

In one-at-a-time mode (.IN bit set), when the processor finds a mismatch between corresponding input and reference bits, the processor:

- Stops the search and sets the .FD bit so the ladder program can take appropriate action.
- Records the bit position in the result file

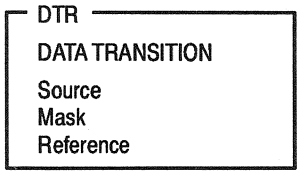
To continue the compare operation, you must re-enable the rung.

In all-per-scan mode (.IN bit reset), the processor detects all mismatches between corresponding input and reference bits in one program scan. Then the processor:

- Writes the bit positions of mismatched bits into the result file in the same order the processor detects the mismatches
- Sets the .FD and .DN bits so your ladder program can take appropriate action.

Important: The DDT instruction also changes the status of the mismatched reference bit to the same state as the the source bit each time it finds a mismatch.

Data Transitional (DTR)



The DTR instruction is an input instruction that you use to pass a source value through a mask and compare the result to a reference value. You can use this instruction for detecting and identifying invalid inputs and for preventing invalid inputs from shutting down a batch processor or machine operation.

Description:
The DTR instruction compares a source word through a mask with a reference word. The instruction also writes the source word into the reference address for the next comparison. The source word remains unchanged.

When the masked source differs from the reference, the instruction goes true for only one scan. The processor writes the masked source value into the reference address. When the masked source and the reference are the same, the instruction remains false.



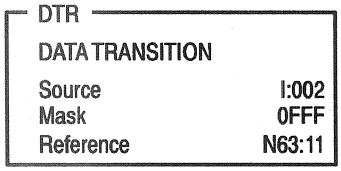
CAUTION: Online programming with this instruction can be dangerous. If the destination value is different from the source value, the instruction goes true. Use caution if you insert this instruction when the processor is in Run or Remote Run mode.

Entering Parameters

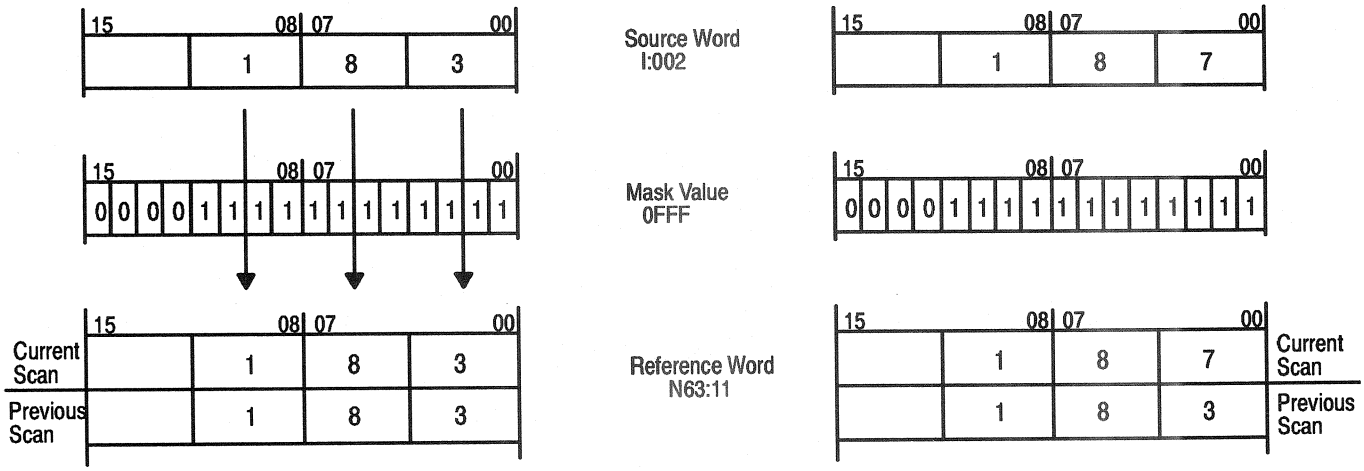
To program the DTR instruction, you need to provide the processor with the following information:

Parameter:	Definition:
source	the address of the input word, typically real inputs.
mask	the hexadecimal value or address that contains the mask value
reference	the address of the reference word The reference contains the source data from the last DTR scan

Data Transition Example:



For example, the DTR instruction above passes the source (I:002) through a mask of OFFF and compares the result to the reference file (N63:11). The source word is then written into the reference address for the next comparison (the source remains unchanged).



Rung remains false as long as input value does not change

Rung goes true for one scan when change is detected

13385

Chapter 11

Shift Register Instructions BSL, BSR, FFL, FFU, LFL, LFU

Applying Shift Registers

A shift register is often used to simulate the movement or flow of parts and information.

If You Use a Shift Register for:	Data in the Shift Register Could Represent:
Tracking parts through an assembly line	Part types, quality, size, and status
Controlling machine or process operations	The order in which events occur
Inventory control	Identification numbers or locations
System diagnostics	A fault condition that caused a shutdown

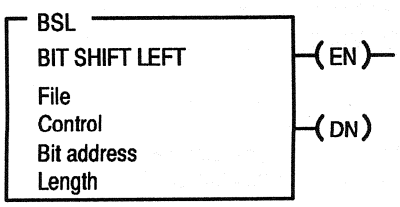
Table 11.A lists the available shift instructions.

Table 11.A
Available Shift Instructions

If you want to	Use These Instructions:	Found on Page:
Load bits into, shift bits through, and unload bits from a bit array one bit at a time, such as for tracking bottles through a bottling line where each bit represents a bottle	BSL, BSR	11-2
Load and unload values in the same order, such as for tracking parts through an assembly line where parts are represented by values that have a part number and assembly code	FFL, FFU	11-5
Load and unload values in reverse order, such as tracking stacked inventory in a warehouse, where goods are represented by serial number and inventory codes	LFL, LFU *	11-8

* These instructions are only supported by PLC-5/30, -5/40 and -5/60 processors.

Using Bit Shift Instructions



Description:
Bit shift instructions shift all bits within the specified address one bit position with each false-to-true rung transition. These instructions are:

- Bit Shift Left (BSL)
- Bit Shift Right (BSR)

Entering Parameters

To program a bit shift instruction, you need to provide the processor with the following information:

- **File** is the address of the bit array you want to manipulate.

  You must start the array at a 16-bit word boundary. For example, use bit 0 of word number 1, 2, 3, etc. You can end the array at any bit number up to 15,999. However, you cannot use the remaining bits in that particular element because the instruction invalidates them.

- **Control** is the address of the control structure (48 bits – 3 16-bit words) in the control area (R) of memory that stores the instruction's status bits, the size of the array (number of bits), and the bit pointer.

  Use the control address with mnemonic when you address these parameters:

- **Position (.POS)** toggles bewteen 0 and a value equal to the length of the bit file.



WARNING: Do not use the same control address for more than one instruction. Unexpected operation could result in possible equipment damage and/or personal injury.

- **Bit Address** is the address of the source bit. The instruction inserts the status of this bit in either the first (lowest) bit position (for the BSL instruction) or the last (highest) bit position (for the BSR instruction) in the array.
- **Length** is the number of bits in the bit array, from 1 to 16,000 bits. Address the length by mnemonic .LEN.

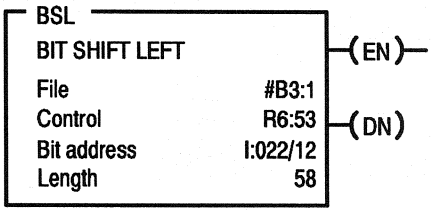
Using Status Bits

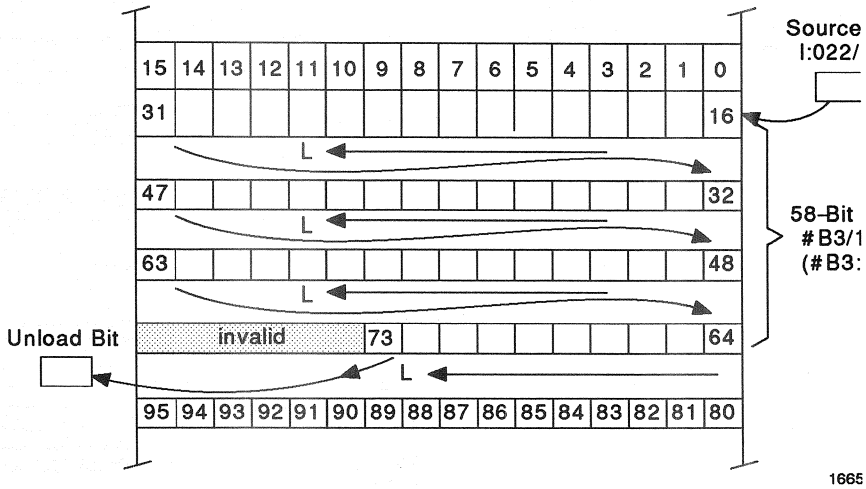
To use the BSL or BSR instruction correctly, the ladder program must examine status bits in the control element. You address these bits by mnemonic.

Bit:	Definition:
Enable .EN (bit 15)	is set when the rung makes a false-to-true rung transition to indicate the instruction is enabled.
Done .DN (bit 13)	is set to indicate that the bit array shifted one bit position
Error .ER (bit 11)	is set to indicate that the instruction detected an error, such as if you entered a negative file length
Unload .UL (bit 10)	is the instruction's output. The .UL bit stores the status of the bit removed from the array each time the instruction is enabled. Avoid using the .UL bit when the .ER bit is set.

Important: When enabled, the bit pointer is set to the value of the length the bit array is shifted. After all of the bits are shifted, the instruction resets the .EN, .ER and .DN bits and the bit pointer when input conditions go false.

Bit Shift Left (BSL) Example:





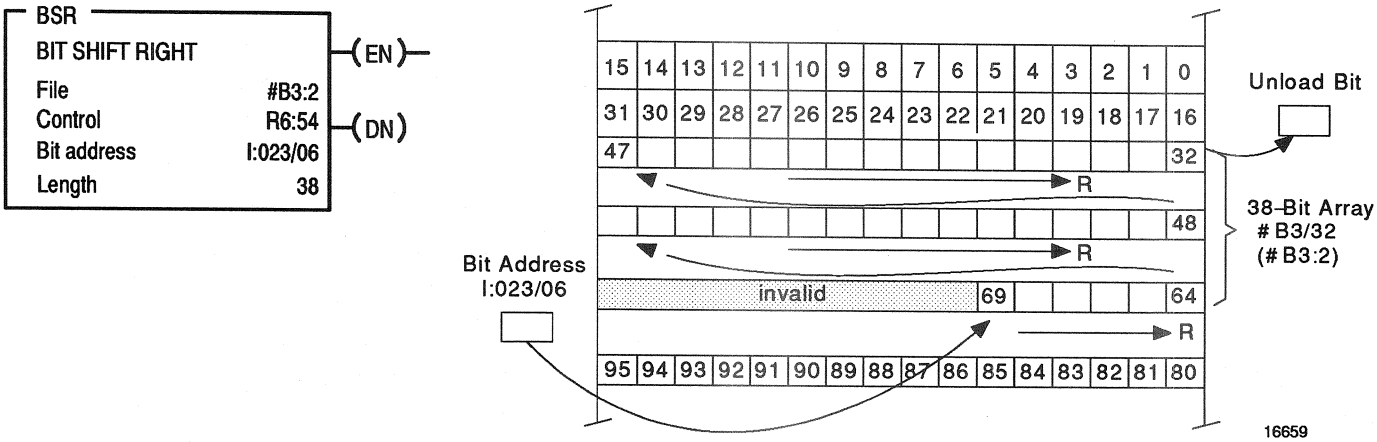
This Parameter:	Tells the Processor:
File (#B3:1)	The location of the bit array
Control (R6:53)	The instruction's address and control element
Bit Address (I:022/12)	The location of the source bit (bit 12 of input word 22)
Length (58)	The number of bits in the bit array

When a rung containing the BSL instruction goes from false to true, the processor sets the .EN bit. Then the processor shifts 58 bits in bit file B3, starting with bit 16, to the left (higher bit number) one bit position. The last bit shifts out at bit position 73 into the .UL bit. The specified source bit, bit 12 of input word 22, shifts into the first bit position, bit 16 of bit file B3.

The processor completes the shift operation in one program scan and sets the .DN bit. Then when the rung goes false, the instruction resets the .EN, .ER (if set) and .DN bits, and resets the pointer.

For wrap-around operation, make the source address the same as the highest (outgoing) bit address. You can omit using the .UL bit in wrap-around operation.

Bit Shift Right (BSR) Example:



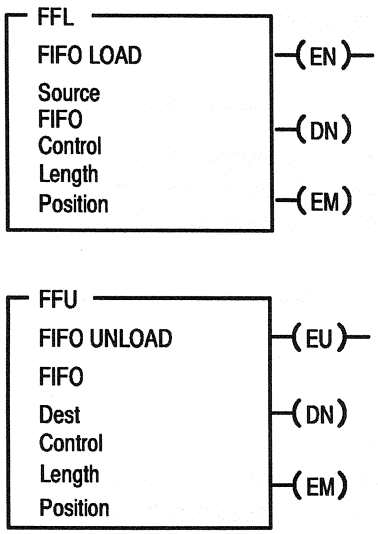
This Parameter:	Tells the Processor:
File (#B3:2)	The location of the bit array
Control (R6:54)	The instruction's address and control element
Bit Address (I:023/06)	The source bit address (bit 06 in input word 23)
Length (38)	The number of bits in the bit array

When a rung containing the BSR instruction goes from false to true, the processor sets the .EN bit. Then the processor shifts 38 bits in bit file B3 to the right (to a lower bit number) one bit position starting with the highest bit position 69. The lowest bit (bit 32) shifts out of the bit array into the .UL bit. The specified source, bit 06 of input word 23, shifts into the highest bit position 69.

The processor completes the operation in one program scan and sets the .DN bit. Then when the rung goes false, the instruction resets the .EN, .ER (if set) and .DN bits, and resets the pointer.

For wrap-around operation, make the source address the same as the lowest (outgoing) bit address. You can omit using the .UL bit in wrap-around operation.

Using FIFO and LIFO Instructions



Description

Use FIFO instructions, First In – First Out (FFL and FFU) and LIFO instructions, Last In – First Out (LFL and LFU) in pairs to store and retrieve data in a prescribed order.

These Instructions:	Retrieve Data:
FFL and FFU	In the order stored (first in, first out)
LFL and LFU *	In reverse of the order stored (last in, first out)
* Available on PLC-5/30, -5/40 and -5/60 processors only	

When used in pairs, these instructions establish an asynchronous shift register (stack).

Entering Parameters

When you program a FIFO or LIFO stack, use the same file and control addresses, length, and position values for both instructions in the pair. You need to provide the processor with the following information:

- **Source** is the address that stores the "next in" value to the stack. The FIFO or LIFO load instruction (FFL or LFL) retrieves the value from this address and loads it into the next word in the stack.
- **Destination** is the address that stores the value that exits from the stack.

This Instruction:	Unloads the Value from:
FIFO's FFU	Word zero
LIFO's LFU	The last word entered

- **FIFO** or **LIFO** is an indexed address of the stack. Use the same FIFO address for the associated FFL and FFU instructions; use the same LIFO address for the associated LFL and LFU instructions.
- **Control** is the address of the control structure (48 bits – 3 16-bit words) in the control area (R) of memory. The control structure stores the instruction's status bits, stack length, and next available position (pointer) in the stack.

  Use the control address with mnemonic when you address the following parameters:

  - **Length (.LEN)** is the maximum number of elements in the stack.
  - **Position (.POS)** indicates the next available location where the instruction loads data into the stack.

- **Length** lets you specify the maximum number of words in the stack. Address the length value by mnemonic .LEN.

- **Position** indicates the next available location where the instruction loads data into the stack. Address the position value by mnemonic .POS.

  Enter a position value only if you want the instruction to start at an offset at power-up. Otherwise, enter 0. Your ladder program can change the position if necessary.



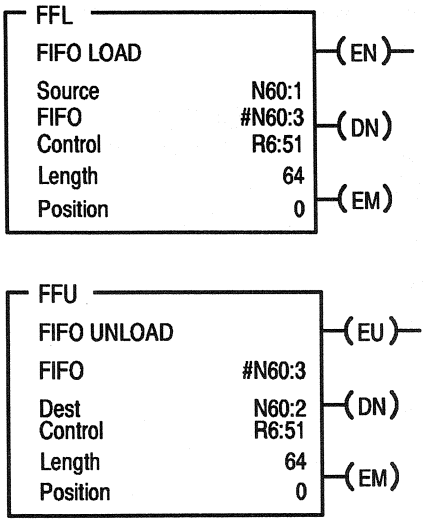
WARNING: Except when pairing stack instructions, do not use the same control address for any other instruction. Unexpected operation could result with possible equipment damage and/or personal injury.

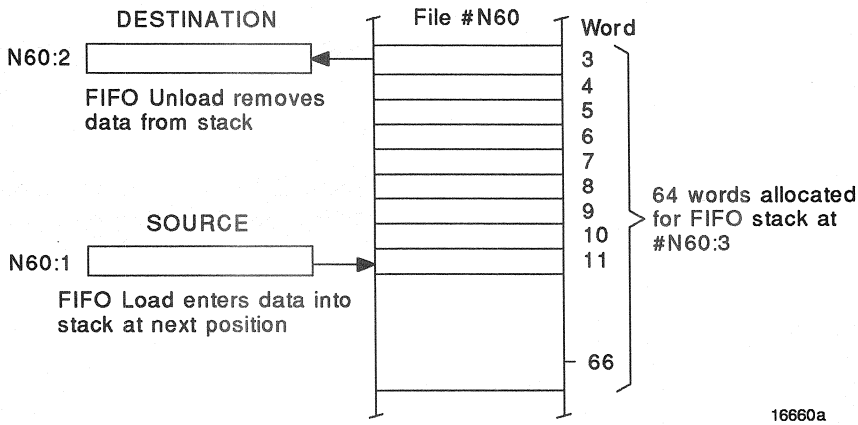
Using Status Bits

To use the FIFO and LIFO instructions correctly, the ladder program must examine status bits in the control structure. You address these bits by mnemonic.

This Bit:	Is Set:
Enable Load .EN (bit 15)	when the rung makes a false-to-true rung transition to indicate the instruction is enabled (used in FFL and LFL instructions).
Enable Unload .EU (bit 14)	when rung conditions are true to indicate the instruction is enabled (used in FFU and LFU instructions).
Done .DN (bit 13)	by the processor to indicate that the stack is full. The .DN bit inhibits loading the stack until there is room.
Empty .EM (bit 12)	by the processor to indicate that the stack is empty. Do not enable the FIFO or LIFO unload commands if the .EM bit is set.

FIFO Load (FFL) and FIFO Unload (FFU) Example:





This Parameter:	Tells the Processor:
Source (N60:1)	The location of the "next in" source word
FIFO (#N60:3)	The location of the stack (FIFO file)
Dest (N60:2)	The location of the "exit" word
Control (R6:51)	The instruction's address and control structure
Length (64)	The maximum number of words you can load
Position (0)	To start at the FIFO file address

FIFO Load Description:

When the rung that contains the FFL instruction goes from false to true, the processor sets the .EN bit and loads the source element (N60:1) into the next available element in the stack as pointed to by the control structure's position. The processor loads an element each time the rung goes from false to true, until it fills the stack. When the stack becomes full, the processor sets the .DN bit. Your ladder program should detect that the stack is full and inhibit further loading of data from the source.

You may want to load the stack in advance, or enable the load instruction while inhibiting the unload instruction until the stack contains the desired data.

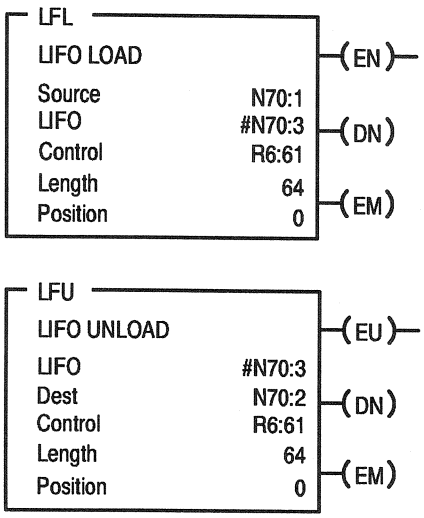
FIFO Unload Description:

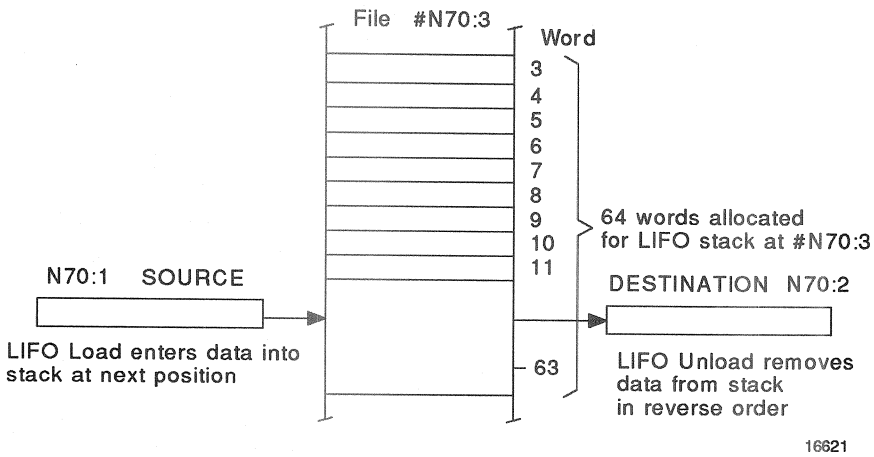
When the rung that contains the FFU instruction goes from false to true, the processor sets the .EU bit and unloads data from the first element stored in the FIFO stack into the destination word N60:2. At the same time the processor shifts all data in the stack one position toward the first word. The processor unloads one word each time the rung goes from false to true until it empties the FIFO stack.

When the stack becomes empty, the processor sets the .EM bit. Thereafter, the processor transfers a zero value for each false-to-true rung transition until the FFL instruction loads new values. Your ladder program should detect that the stack is empty and inhibit other instructions from using zero values stored at the destination.

With a FFU instruction, you can unload data from a word other than the first word of the stack by changing the FIFO address to the address of the desired word and changing the length accordingly.

LIFO Load (LFL) and LIFO Unload (LFU) Example: (PLC-5/30, -5/40 and -5/60 processors only)





This Parameter:	Tells the Processor:
Source (N70:1)	the location of the "next in" source word
LIFO (#N70:3)	the location of the stack (LIFO file)
Dest (N70:2)	the location of the "exit" word
Control (R6:61)	the instruction's control structure
Length (64)	the maximum number of words you can load
Position (0)	to start at the LIFO file address

Important: The difference between FIFO and LIFO stack operation is that the LFU instruction removes data in the **reverse** order to the order it is loaded (last-in-first-out). Otherwise, LIFO instructions operate identical to FIFO instructions.

LIFO Load Description:
When the rung that contains the LFL instruction goes from false to true, the processor sets the .EN bit and loads the source word (N70:1) into the next available word in the stack as pointed to by the control structure's position. The processor loads an element each time the rung goes from false to true until it fills the stack. When the stack becomes full, the processor sets the done bit. Your ladder program should detect that the stack is full and inhibit further loading of data from the source.

You may want to load the stack in advance, or enable the load instruction while inhibiting the unload instruction until the stack contains the desired data.

LIFO Unload Description:
When the rung that contains the LFU instruction goes from false to true, the processor sets the .EU bit and unloads data starting with the last word stored in the LIFO stack into the destination word N70:2. The processor unloads one word each time the rung goes from false to true until it empties the LIFO stack.

When the stack becomes empty, the processor sets the .EM bit. Thereafter, the processor transfers a zero value for each false-to-true rung transition until the load instruction loads new values. Your ladder program should detect that the stack is empty and inhibit other instructions from using zero values stored at the destination.

With a LIFO unload instruction, you can unload data from a word other than the first word of the stack by changing the LIFO address to the address of the desired word and changing the length accordingly.

Chapter 12

Sequencer Instructions SQI, SQL, SQO

Applying Sequencers

Sequencer instructions are used typically to control automatic assembly machines that have a consistent and repeatable operation. Use the sequencer input instruction to detect when a step is complete; use the sequencer output instruction to set output conditions for the next step. Use the sequencer load instruction to load reference conditions into the sequencer input and output file.

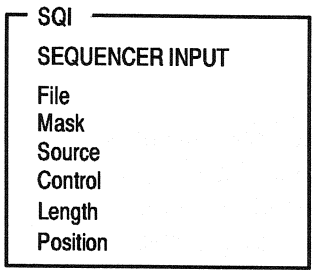
Table 12.A lists the available sequencer instructions.

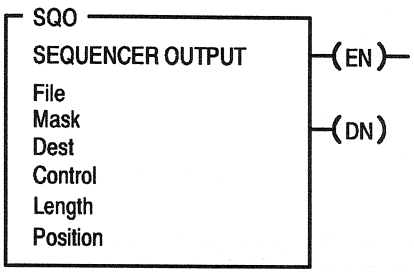
Table 12.A
Available Sequencer Instructions

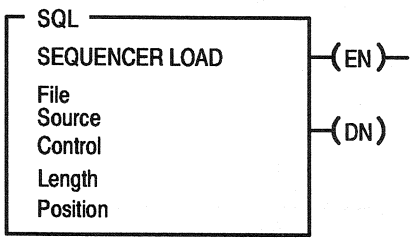
If You Want to:	Use This Instruction:	Found on Page:
Control sequential machine operations by transferring 16-bit data through a mask to output image addresses	SQO	12-4
Monitor machine operating conditions for diagnostic purposes by comparing 16-bit image data (through a mask) with data in a reference file	SQI	12-6
Capture reference conditions by manually stepping the machine through its operating sequences and loading I/O or storage data into destination files	SQL	12-8

The primary advantage of sequencer instructions is to conserve program memory. These instructions monitor and control multiples of 16 discrete outputs at a time in a single rung.

Using Sequencer Instructions







Description:
Use the sequencer input (SQI) and sequencer output (SQO) instructions in pairs to respectively monitor and control a sequential operation. Use the sequencer load (SQL) instruction to load data in the sequencer file.

These instructions operate on multiples of 16 bits at a time. Place SQI instructions in series and SQO instructions in parallel in the same rung for 32-, 48-, 64-, or other bit operations.

Important: Each SQO instruction on a branch increments the control structure, so corresponding SQI instructions may miss parts of the source file.

Entering Parameters

When programming sequencer input and output instructions in pairs, use the same control address, length value, and position value in each instruction. The same applies when using multiple instructions in the same rung to double, triple, or further increase the number of bits.

To program sequencer instructions, you need to provide the processor with the following information:

- **File** is the indexed address of the sequencer file to or from which the instruction transfers data. Its purpose depends on the instruction:

In This Instruction:	The Sequencer File Stores Data for:
SQO	Controlling outputs
SQI	Reference to detect completion of a step or a fault condition
SQL	Creating the SQO or SQI file

- **Mask** (for SQO and SQI) is a hexadecimal code or the address of the mask element or file through which the instruction moves data. Set (1) mask bits to pass data; reset (0) mask bits to prevent the instruction from operating on corresponding destination bits. Specify a hexadecimal value for a constant mask value. Store the mask in an element or file if you want to change the mask according to application requirements.
- **Source** (for SQI and SQL) is the address of the input element or file from which the instruction obtains data for its sequencer file.
- **Destination** (for SQO, only) is the destination address of the output word or file to which the instruction moves data from its sequencer file.

Important: If you use a file for the source, mask, or destination of a sequencer instruction, the instruction automatically determines the file length and moves through the file step-by-step as it moves through the sequencer file.

- **Control** is the address of the control structure in the control area (R) of memory (48 bits – 3 16-bit words) that stores the instruction's status bits, the length of the sequencer file, and the instantaneous position in the file.

  Use the control address with mnemonic when you address the following parameters:

  - **Length (.LEN)** is the length of the sequencer file.
  - **Position (.POS)** is the current position of the word in the sequencer file that the processor is using.

For This Instruction:	The Control Structure Is Incremented:
SQO and SQL	By the instruction itself
SQI	Externally, either by the paired SQO with the same control address, or by another instruction



WARNING: Except for paired instructions, do not use the same control address for any other purpose. Duplication of a control element could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

- **Length** is the number of steps of the sequencer file starting at position 1. Position 0 is the start-up position. The instruction resets to position 1 at each completion.

Important: The address assigned for a sequencer file is step zero. Sequencer instructions use (length + 1) words of data for each file referenced in the instruction. This also applies to the source, mask, and destination values if addressed as files.

- **Position** is the word location in the sequencer file. The position value is incremented internally by SQO and SQL instructions.

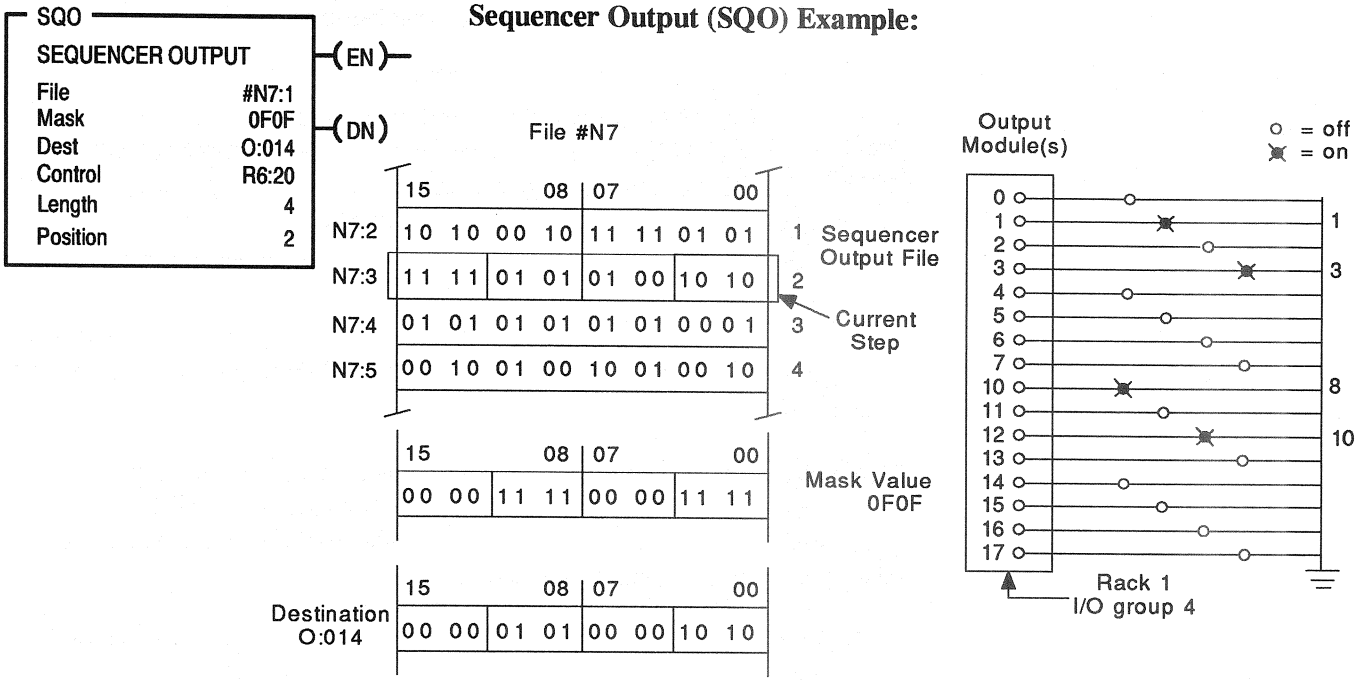
Important: Your ladder program can externally increment the position value of the SQI instruction. One way to do this is to pair it with the SQO instruction and assign the same control structure to both instructions.

Using Status Bits

To use the sequencer instructions correctly, the ladder program must examine status bits in the control element. You address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	(SQO or SQL) is set on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition.
Done .Dn (bit 13)	(SQO or SQL) is set after the instruction finishes operating on the last word in the sequencer file. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the length value is less than or equal to zero or when the position value is less than zero

Sequencer Output (SQO) Example:



SQO instruction moves output data through a mask to an output word for controlling multiple outputs.

16645a

This Parameter:	Tells the Processor:
File (#N7:1)	The location of the sequencer file
Mask (0F0F)	The fixed hexadecimal value of the mask
Dest (O:014)	The output image address to be changed
Control (R6:20)	The structure that controls the operation
Length (4)	The number of words to step through
Position (2)	The current position

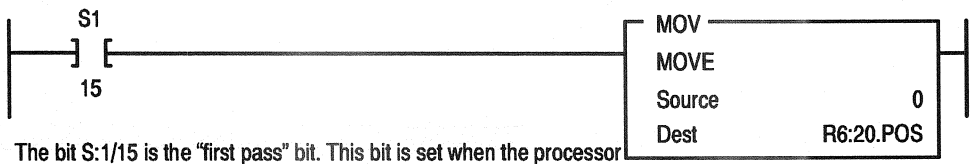
The SQO instruction steps through the sequencer file of 16-bit output words whose bits have been set to control various output devices.

When the rung goes from false to true, the instruction increments to the next step (word) in the sequencer file #N7:1. The data in the sequencer file is transferred through a fixed mask (0F0F) to the destination address O:014. Current data is written to the destination element every scan that the rung remains true.

At start-up when you switch the processor from program mode to run mode, instruction operation depends on whether the rung is true or false on the first scan:

- If the rung is true and POS = 0, the instruction transfers data in step 0.
- If rung is false, the instruction waits for the first false-to-true rung transition and transfers data in step 1.

After transferring the last word of the sequencer file, the processor sets the .DN bit. On the next false-to-true rung transition, the processor resets the .DN bit and resets the position to step 1.

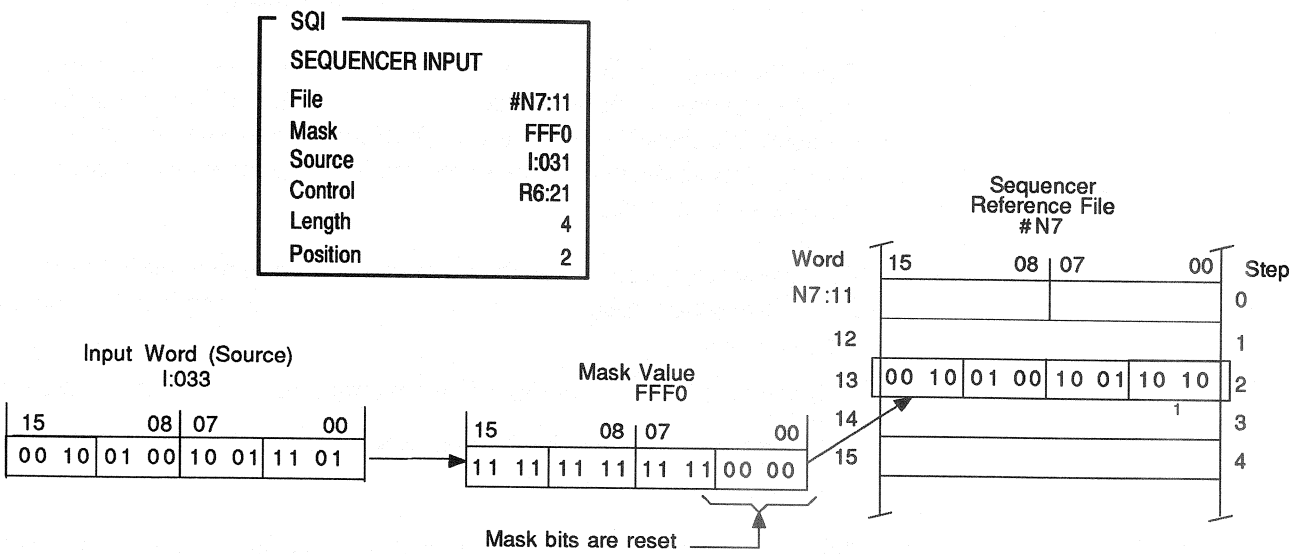


The bit S:1/15 is the "first pass" bit. This bit is set when the processor first scans a program. When this rung goes true, the processor moves the value of 0 to the position word of the SQO instruction. On the next false-to-true rung transition, the position says to start at step 0 and the actual step the processor runs is step 1.

Resetting the Position of SQO

Each time the processor goes from Program to Run mode, you should reset the position of any SQO instruction. To do this, use the following ladder logic:

Sequencer Input (SQI) Example:



SQI instruction is true when it detects that an input word matches (through a mask) its corresponding reference word.

[1] These bits are not compared. Therefore, the instruction is true in this example.

16646a

This Parameter:	Tells the Processor:
File (#N7:11)	The location of the reference file
Mask (FFF0)	The fixed hexadecimal value of the mask
Source (#I:031)	The input image address to be compared
Control (R6:21)	The element that controls the operation
Length (4)	The number of elements to step through
Position (2)	The current position

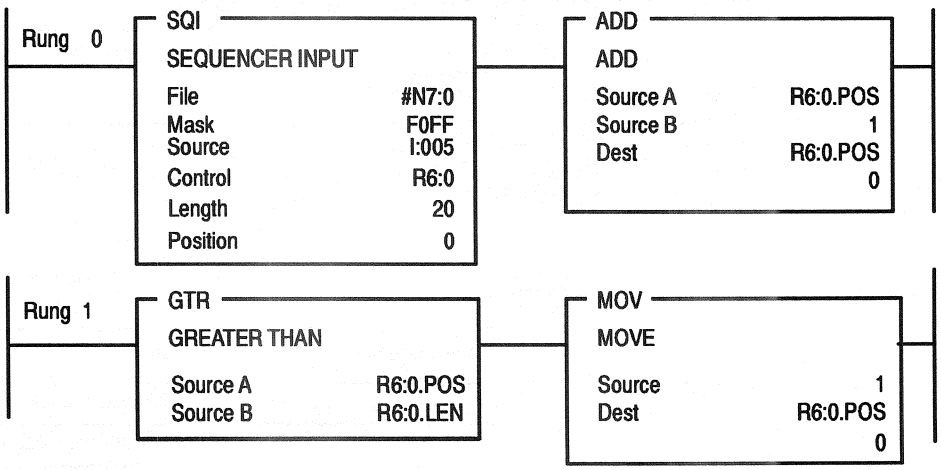
The SQI instruction compares a file of input image data (I:031), through a mask (FFF0), to a file of reference data (N7:11) for equality. When the status of all non-masked bits of the word at that particular step match those of the corresponding reference word, the instruction goes true. Otherwise, the instruction is false.

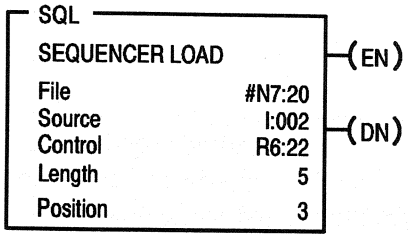
Important: You can use the SQI instruction with the control structure of the SQO instruction. Program the SQI as the condition instruction in the same rung with the SQO. Assign the same control address and length to both instructions so they track together.

Using SQI Without SQO

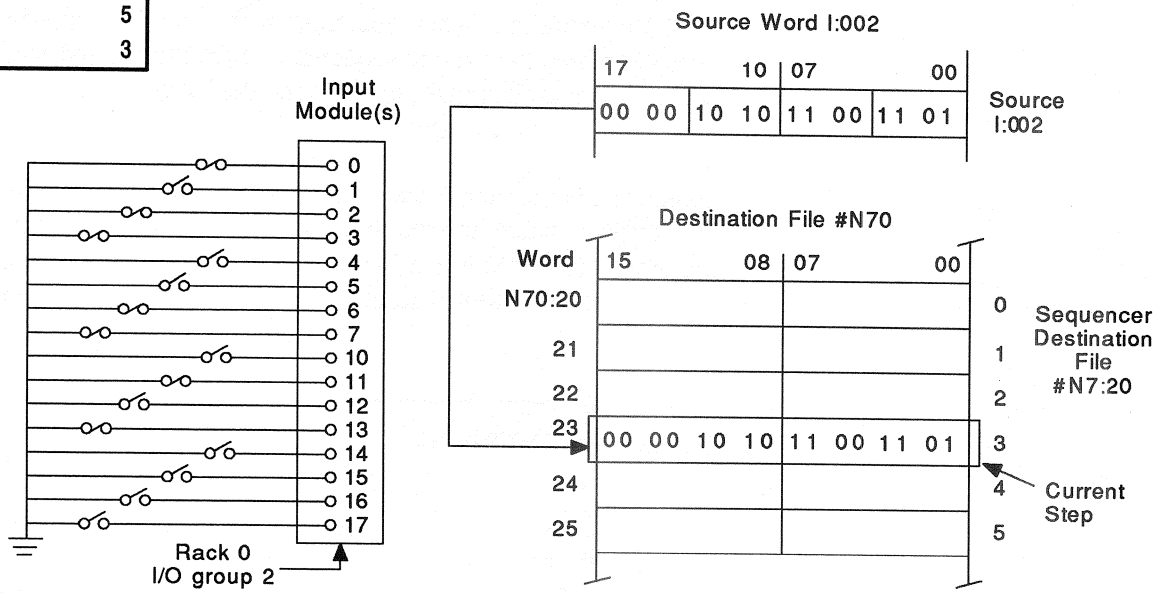
Another application of the SQI instruction is machine diagnostics where you load the reference file with data representing the desired sequence of machine operation. When operating, if the real time sequence of operation does not match the desired sequence of operation stored in the reference file, enable a fault signal. In this case, the ladder program externally increments the SQI instruction.

To externally increment the sequencer file, use a CPT instruction to move a new position value into the SQI instruction's control element. Do this to increment each step in the SQI instruction's file. Rung 0 increments the SQI instruction. Rung 1 resets the position value after stepping through the file.





Sequencer Load (SQL) Example:



SQL instruction loads data from the input word into a destination file from where it can be moved to other (sequencer) files. 16661a

This Parameter:	Tells the Processor:
File (#N7:20)	The location of the destination file
Source (I:002)	The input image address to be read
Control (R6:22)	The structure that controls the operation
Length (5)	The number of words to step through
Position (3)	The current step

When the rung goes from false to true, the SQL instruction increments to the next step in the sequencer file and loads data into it, one step for each rung transition. The SQL instruction loads current data each scan that the rung remains true. A mask is not used.

At start-up when you switch the processor from Program to Run mode, the instruction operation depends on whether the rung is true or false on the first scan:

- If the rung is true, the instruction loads data into step 0.
- If the rung is false, the instruction waits for the first false-to-true rung transition and loads data into step 1.

After loading the last step, the processor sets the .DN bit. On the next false-to-true transition, the processor resets its .DN bit, resets the position to step 1, and loads data into that word.

Chapter 13

Program Control Instructions MCR, JMP, LBL, FOR, NXT, BRK, JSR, SBR, RET, TND, AFI, ONS, OSR, OSF, SFR, EOT, UIE, UID

Selecting Program Flow Instructions

Program flow instructions change the flow of ladder program execution. Use Table 13.A to select the program control instruction or group of instructions that fit your programming requirements.

Table 13.A
Available Program Control Instructions

If You Want to:	Then Use These Instructions:	Found on Page:
Turn off all non-retentive outputs in a section of ladder program	MCR	13-2
Jump over a section of a program that does not always need to be executed	JMP, LBL	13-3
Loop through a set of rungs for a preset number of times	FOR, NXT, BRK, LBL, RET	13-5
Jump to a separate subroutine file, pass data to the subroutine, perform an operation, and return the results	JSR, SBR, RET	13-8
Mark a temporary edit that halts program execution beyond it	TND	13-14
Disable a rung	AFI	13-14
Trigger a one-shot event based on a change in rung condition	ONS, OSR,* OSF*	13-14
Reset a sequential function chart	SFR*	13-16
End a transition file	EOT	13-17
Enable or disable user interrupts	UIE,* UID*	13-18

* These instructions are only supported by PLC-5/30, -5/40 and -5/60 processors.

Master Control Reset (MCR)

–(MCR)–

Description:
Use MCR instructions in pairs to create program zones that turn off all the non-retentive outputs in the zone. Non-retentive outputs are reset when their rung goes false.

If the MCR Rung That Starts the Zone Is:	Then the Processor:
True	Executes the rungs in the MCR zone based on each rung's individual input conditions (as if the zone did not exist)
False	Resets all non-retentive output instructions in the MCR zone regardless of each rung's individual input conditions.

MCR zones let you enable or inhibit segments of your program, such as for recipe applications.

When you program MCR instructions, note that:

- You end the zone with an unconditional MCR instruction.
- You cannot nest one MCR zone within another.
- Do not jump into an MCR zone. If the zone is false, jumping into it activates the zone.
- If an MCR zone continues to the end of the ladder program, you do not have to program an MCR instruction to end the zone.

Important: The MCR instruction is not a substitute for a hard-wired master control relay that provides emergency stop capability. You still should install a hard-wired master control relay to provide emergency I/O power shutdown.

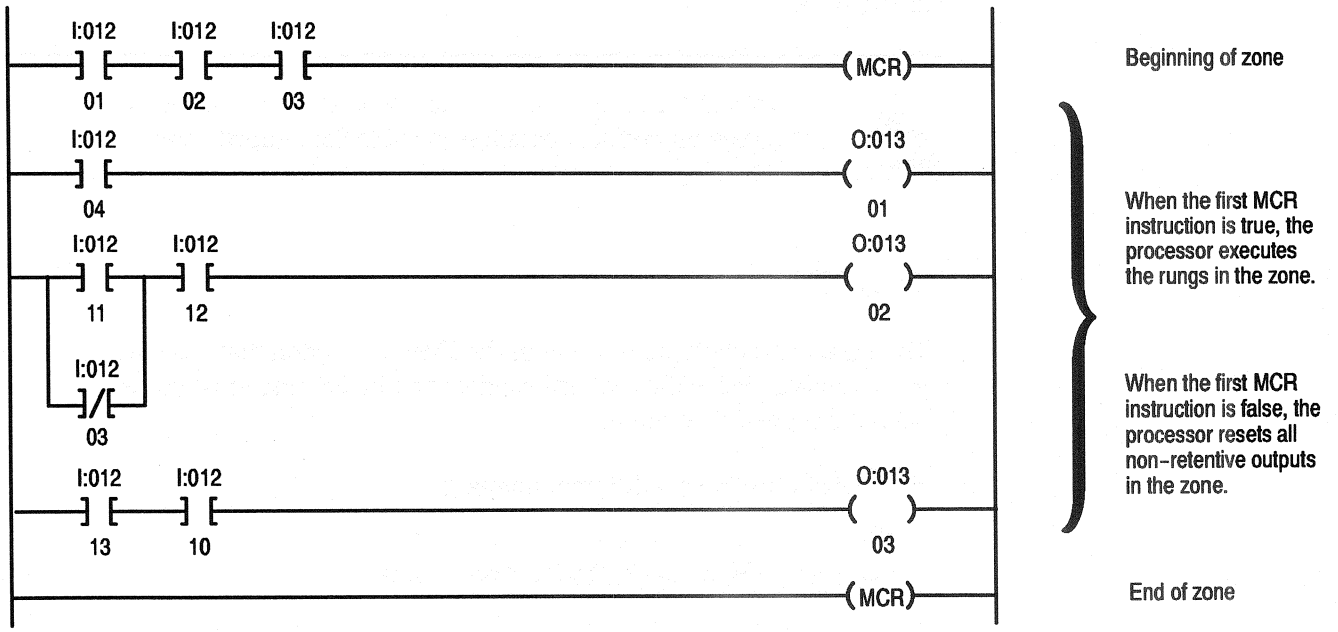


WARNING: Do not overlap or nest MCR zones. Each MCR zone must be separate and complete. If overlapped or nested, unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

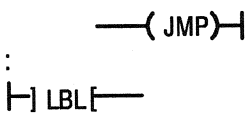


WARNING: If you start instructions such as timers or counters in an MCR zone, instruction operation ceases when the zone is disabled. Re-program critical operations outside the zone if necessary.

Example:
When the rung containing the first MCR instruction is true, the processor executes the rungs in the MCR zone based on the rung input conditions. Otherwise, the processor resets the non-retentive output instructions within the MCR zone.



Jump (JMP) and Label (LBL)



Description:
You use jump and label instructions in pairs to skip portions of the ladder program.

If the Jump Rung Is:	Then the Processor:
True	Skips from the JMP rung to the LBL rung and continues executing the program. You can jump forward or backward.
False	Ignores the JMP instruction

Jumping forward to a label saves program scan time by omitting a program segment until needed. Jumping backward lets the processor repeat iterations through a program segment until its logic is complete.

Important: Be careful not to jump backwards an excessive number of times. The watchdog timer could time out, which would fault the processor.

Using JMP

The JMP instruction lets the processor skip rungs. Use a 2-digit number (00-31) on PLC-5/10, -5/12, -5/15, -5/25 processors to identify the jump instruction. Use a 3-digit number (000-255) on PLC-5/30, -5/40 or -5/60 processors. You can jump to the same label from one or more JMP instructions.



WARNING: Jumped timers and counters are not scanned. Re-program critical operations outside the jumped zone.

Using LBL

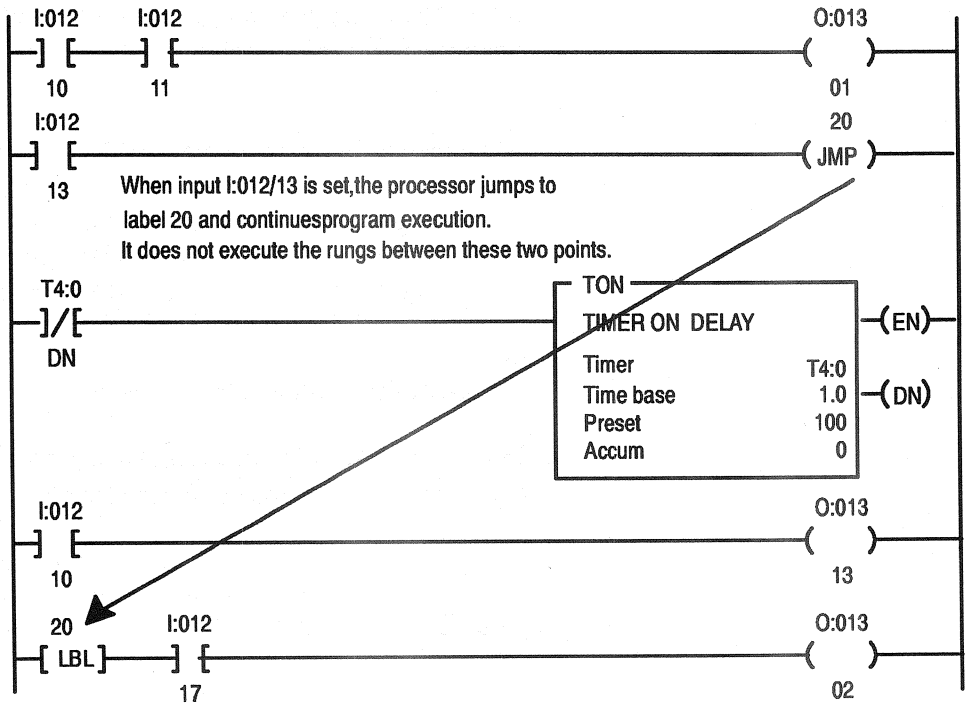
The LBL instruction is the target of the JMP instruction that has the same label number. Place the LBL instruction first in the rung to where you want the processor to jump.

Each label must have a different number:

- PLC-5/30, -5/40 and -5/60 processors: 000-255
- PLC-5/10, -5/12, -5/15, -5/25 processors: 0-31

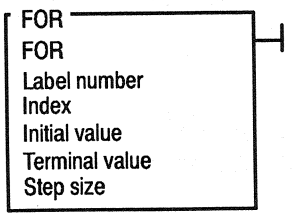
You can place up to 256 labels in one PLC-5/30, -5/40 or -5/60 program file; you can place up to 32 labels in one PLC-5/10, -5/12, -5/15, -5/25 program file.

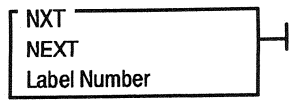
JMP and LBL Example:
When the rung containing the JMP instruction goes true, the processor jumps over successive rungs until it reaches the rung that contains the LBL instruction with the same number. The processor resumes executing at the LBL rung.



The timer (TON) will not update as long as I:012/13 is true.

For Next Loop (FOR, NXT), Break (BRK)





Description:
Use FOR, BRK and NXT instructions to create your own programming routines where you control the number of times the loop is executed.

Entering Parameters

To program the FOR instruction, you must provide the processor with the following information:

- **Label number** is the unique label number that marks the location of the FOR instruction. Enter a unique number. The PLC-5/10, -5/12, -5/15, -5/25 processor support 0–31 labels; PLC-5/30, -5/40 and -5/60 processors support 0–255 labels.
- **Index** is the logical address where the instruction stores the index value it computes.

The index value is the sum of the initial value plus the accumulating step values. The FOR instruction uses the index value to determine the number of times the loop is executed.

When you enable the FOR instruction, the processor sets the index value equal to the initial value. Then , if the index values is less than or equal to the final value, the processor steps to the following instruction. If the index is greater than the fixed value, the processor jumps to the NXT instruction.

When the processor encounters a NXT instruction, it compares the index with the final value. If the index is less than or equal to the final value, the processor jumps back to the FOR instruction. Otherwise it steps to the following instruction. If the processor encounters a BRK on a true rung, the processor skips to the instruction following the NXT.

- **Initial value** (index value) is an integer value or integer address that represents the starting value for the loop.
- **Terminal value** (reference value) is an integer value or integer address that represents the ending value for the loop.
- **Step Size** (constant) is an integer value that specifies the amount to increment the index value. You can change the step value from the ladder program.

Using FOR

When the rung is true, the FOR instruction executes the rungs between the FOR and NXT repeatedly in one program scan until it reaches the preset number of loops, or a BRK instruction aborts the operation. The FOR instruction repeats this operation each scan its rung is true. The FOR instruction does not require a rung transition to begin operation.

Important: Be careful not to loop too many times in the single program scan. An excessive number of calls causes the watchdog timer to time out, which faults the processor.

You can change the initial and terminal values from the main program before executing the FOR instruction. You should not change the index value that the instruction computes.

Also, if you edit a FOR/NXT instruction in Rem RUN mode, make sure that you make the corresponding changes to both rungs before assembling edits. For example, if you want to change the label number for the FOR/NXT pair, change the label in the FOR instruction and in the NXT instruction; then assemble the edits. If you assemble edits after changing only one of the instructions in the FOR/NXT pair, the processor causes a run-time error or watchdog timeout.



WARNING: Changing the index value could cause the instruction to execute the loop an unexpected number of times with possible damage to equipment and/or injury to personnel.

Using BRK

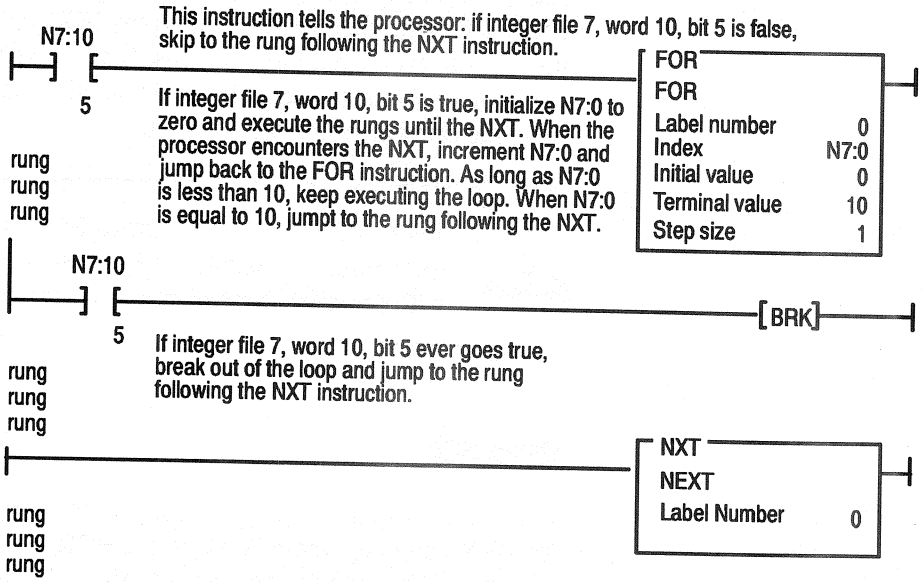
The BRK instruction lets you abort the FOR instruction's operation. Place the BRK rung anywhere between the FOR and NXT. Whenever the rung goes true, it returns the processor to the next highest loop (if you are using nested loops) or to the following instruction after the corresponding NXT instruction in the main program.

Use BRK to exit the loop whenever the processor detects an error or to avoid prolonged loops that could cause the watchdog timer to time out, which would fault the processor.

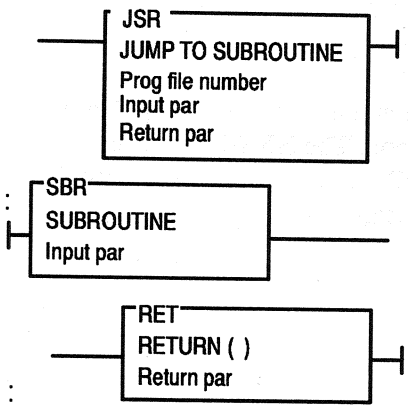
Using NXT

The NXT instruction must be programmed on an unconditional rung that is the last rung to be repeated by the For–Next loop. The NXT instruction returns the processor to the corresponding FOR instruction (identified by the label number specified in the NXT instruction).

FOR, BRK, LBL, and RET Example:



Jump to Subroutine (JSR), Subroutine (SBR), and Return (RET)



Description:

The JSR, SBR, and RET instructions direct the processor to go to a separate subroutine file within the ladder program, scan that subroutine file once, and return to the point of departure.

The JSR instruction directs the processor to the specified subroutine file, and if required, defines the parameters passed to and received from the subroutine. The optional SBR instruction is the header instruction that stores incoming parameters. Use SBR only if you want to pass parameters. The RET instruction ends the subroutine, and if required, stores parameters to be returned to the JSR instruction in the main program.

Important: If you use the SBR instruction, the SBR instruction must be the first instruction on the first rung in the program file that contains the subroutine.

Use a subroutine to store recurring sections of program logic that can be accessed from multiple program files. A subroutine saves memory because you program it only once.

However, update critical I/O within subroutines using immediate input and/or output instructions (IIN, IOT), especially if your application calls for nested or relatively long subroutines. Otherwise, the processor does not update I/O until it reaches the end of the main program (after executing all subroutines). Also note that outputs in subroutines are left in their last state.

Passing Parameters

This subroutine feature lets you pass selected values to a subroutine before execution so the subroutine can perform mathematical or logical operations on the data and return the results to the main program.

For example, you can write a generic subroutine for multiple recipe operations. Then you can pass preset values for each recipe to the subroutine in advance, or have the main program specify and pass presets according to application requirements.

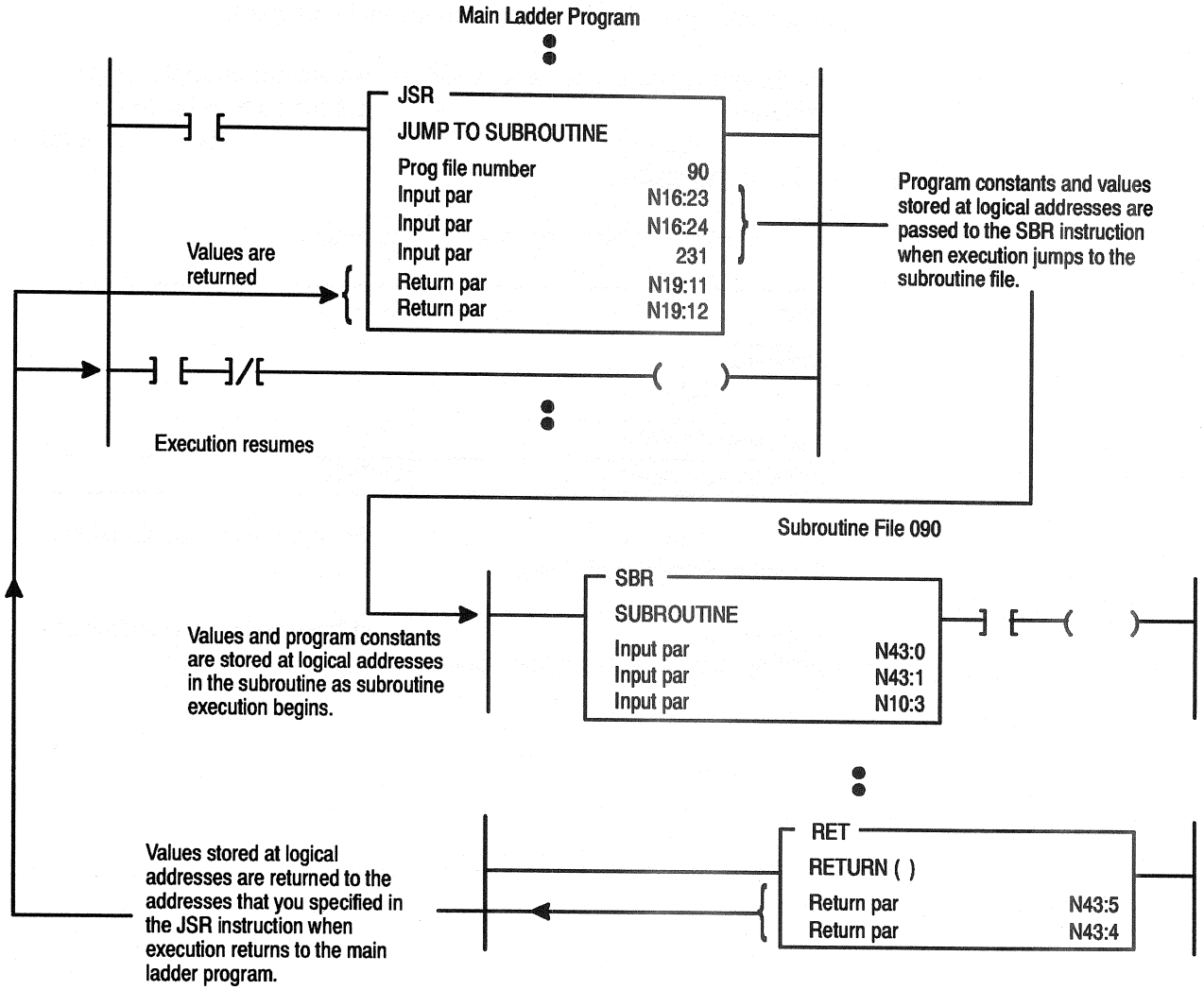
You can pass the following types of parameters:

Type:	Example:
Program constant (integer)	256
Program constant (floating point)	23.467
Logical element address	N7:0
Logical structure address	C5:0.ACC

If you pass floating point data to an integer address, the fractional part of the value is truncated (lost).

Important: Do not mix floating point and integer data and addresses when passing data or you will lose accuracy.

Example of Passing Parameters:
The following diagram shows the passing of parameters between a main program file and a subroutine file.



Entering Parameters

To program these instructions, you must provide the processor with the following information:

Parameter:	Definiiton:
Program file number	the program file number of the file that contains the subroutine.
Input par (JSR)	a program constant or an address of a parameter to be sent to the subroutine
Input par (SBR)	an address where the subroutine stores the incoming data
Return par (JSR)	an address that stores the data received from the subroutine
Return par (RET)	a program constant or an address of a parameter to be returned to the JSR instruction in the main program

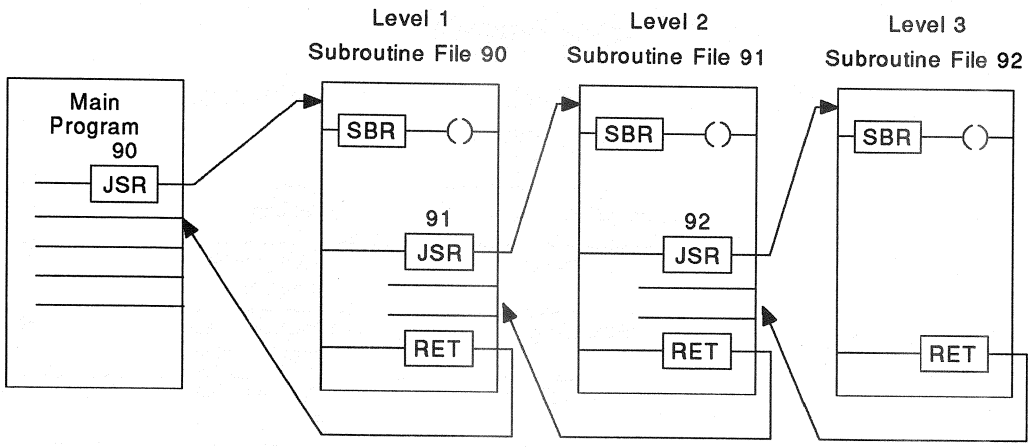
When entering input and return parameters:

- When you enter the JSR instruction, the programming software prompts you for input parameters. After you enter an input parameter, press `[Enter]`. The software prompts you for another input parameter. When you do not have any more input parameters to enter, press `[Enter]`. Then the programming software prompts you for return parameters in the same manner as it did for the input parameters. You cannot enter more than eight input and return parameters combined.
- Make the number of JSR inputs to your subroutine greater than or equal to the number of input parameter addresses in the SBR instruction. Fewer inputs than addresses to receive them causes a run-time error.
- Make the number of RET return parameters greater than or equal to the number of JSR return addresses to receive them. Fewer outputs than addresses to receive them causes a run–time error.

Nesting Subroutine Files

You can nest up to eight subroutines within a program file. This means you can direct program flow from the main program to a subroutine and on to another subroutine, as long there are no more than 7 levels of subroutines.

The path back is the reverse. At RET, the processor automatically returns to the next instruction after the previous JSR instruction. The processor follows this procedure until it returns to the main program.



Using JSR

When a rung that contains a JSR instruction goes true, the processor jumps to a separate subroutine file identified by a file descriptor.

When programming the JSR, keep the following in mind:

- Each subroutine that is external to the main program file must have its own file, identified by a unique file descriptor.
- You cannot jump into any part of the subroutine file except for the first (SBR) instruction in that file.
- You can nest up to eight subroutine files.

Using SBR

When you pass parameters, the SBR instruction must be the first instruction in the first rung of the subroutine. This rung must also have an output instruction. The SBR instruction stores the program constants and data table values passed from the JSR instruction.

Important: If you use the SBR instruction, the SBR instruction must be the first instruction on the first rung in the program file that contains the subroutine.

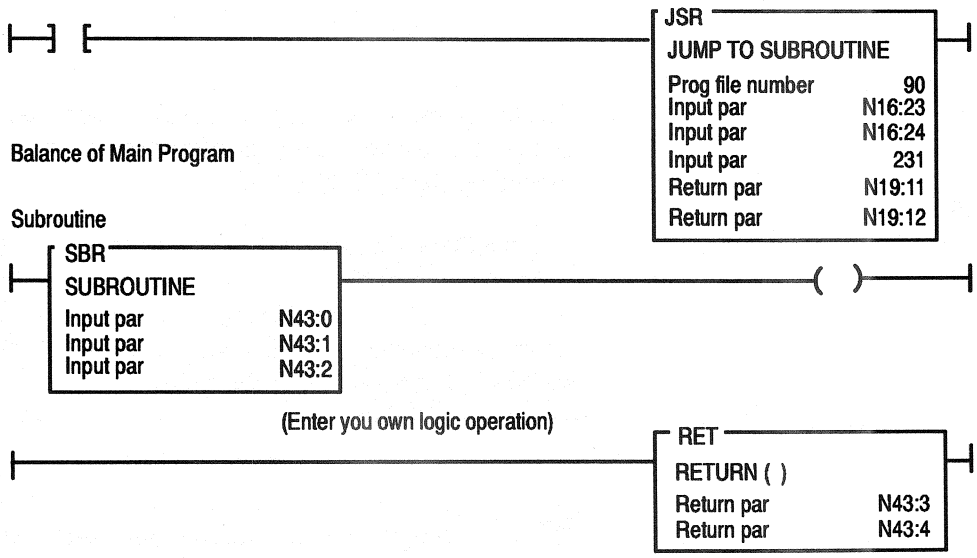
Using RET

The RET instruction concludes subroutine execution. The RET instruction directs the processor back to the instruction following the corresponding JSR instruction. The RET instruction also returns data to the previous subroutine or main program.

Every subroutine must contain an executable RET instruction if you want to return values from the subroutine. The rung that contains the RET instruction can be conditional. If you use this method, you can program the processor to execute only a part of the subroutine if certain conditions are true. However, be sure to program another RET instruction in an unconditional rung at the end of the subroutine to guarantee a valid return from the subroutine when the conditions on the first RET instruction are false.

JSR, SBR, and RET Example:
When the rung that contains the JSR instruction goes true, the processor jumps to the subroutine file specified by the JSR instruction. The processor also passes three values to the subroutine (value stored at N16:23, value stored at N16:24, and constant 231). Then the processor runs the subroutine logic.

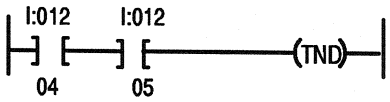
When the processor runs the RET instruction in the subroutine, the processor returns to the instruction following the previous JSR instruction in the main program. The subroutine returns two values to the main program: the value stored at N43:3 is transferred to N19:11, the value stored at N43:4 is transferred to N19:12.



Temporary End (TND)

—(TND)—

Example:



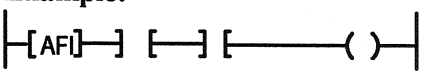
Description:
When the processor encounters the TND instruction, the processor resets the watchdog timer (to zero) performs an I/O update, and begins running the ladder program at the first instruction in the main program.

Insert the TND instruction when debugging or troubleshooting your ladder program. The TND instruction lets your program run only up to this instruction. Move it progressively through your program as you debug each new section. You also use the TND instruction as a boundary between the main program and local subroutines. You can program the TND instruction unconditionally, or condition its rung according to your debugging needs.

Important: Do not confuse the TND instruction with the end-of-program symbol (EOP). You cannot place instructions on the rung that has the EOP symbol.

Always False (AFI)

Example:



Description:
The AFI instruction is an input instruction that makes the rung false when inserted in the condition side of the rung. You can use the AFI instruction to temporarily disable a rung when you debug a program.

One Shot (ONS)



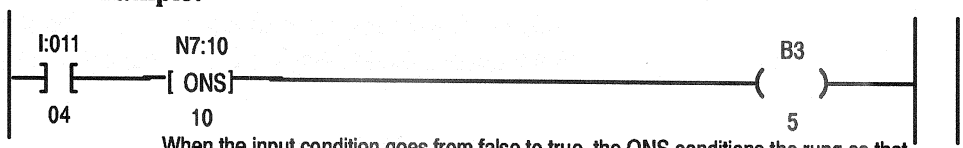
Description:
The ONS instruction is an input instruction that makes the rung true for one program scan upon a false-to-true transitions of the conditions preceding the ONS instruction on the rung.

Use the ONS instruction to start events that are triggered by a pushbutton, such as pulling values from thumbwheel switches or freezing rapidly displayed LED values. You must enter a bit address for the bit. Use either a binary file or integer file address. You can program an output address for the ONS, but be aware of the following:



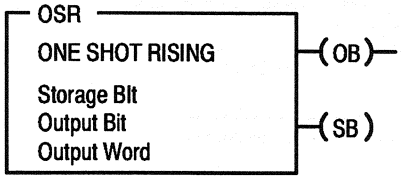
WARNING: On-line programming with this instruction can be dangerous because the output may turn on immediately when the rung is scanned. Set the bit address value to 1 before entering the instruction; then, the rung must go from false to true before energizing its output.

Example:



When the input condition goes from false to true, the ONS conditions the rung so that the output turns on for one scan. The output turns off for successive scans until the input goes from false to true again.

One Shot Rising (OSR) (PLC-5/30, -5/40, -5/60 only)



Description:
The OSR instruction is an output instruction that triggers an event to occur one time. The OSR instruction sets the following bits:

This Bit:	Changes State as Follows:
Output .OB	Is set for one program scan when the rung goes from false to true
Storage .SB	Follows the rung status

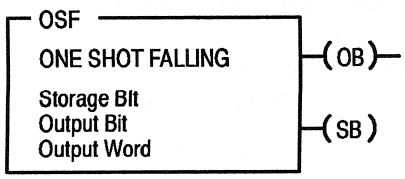
Use the OSR instruction whenever an event must start based on the change of state of the rung from false to true, not on the resulting rung status. You must enter a bit address for the output bit and storage bit. Use either a binary file or integer file address.

Entering Parameters

To program these instructions, you must provide the processor with the following information:

Parameter:	Definition:
Storage bit	the address where you want the storage bit status stored. For example, B3/17
Output bit	the bit position of the address where you want the output bit status stored. For example 5
Output word	the word address where you want the output bit status stored. For example, N7:0

One Shot Falling (OSF) (PLC-5/30, -5/40, -5/60 only)



Description:
The OSF instruction is an output instruction that triggers an event to occur one time. The OSF instruction sets the following bits:

This Bit:	Changes State as Follows:
Output .OB	Is set for one program scan when the rung goes from true to false
Storage .SB	Follows the rung status

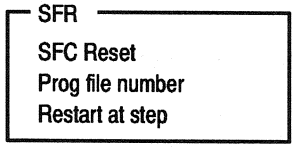
Use the OSF instruction whenever an event must start based on the change of state of the rung from true to false, not on the resulting rung status. You must enter a bit address for the output bit and storage bit. Use either a binary file or integer file address.

Entering Parameters

To program these instructions, you must provide the processor with the following information:

Parameter:	Definition:
Storage bit	the address where you want the storage bit status stored. For example, B3/17.
Output bit	the bit position of the address where you want the output bit status stored. For example 5.
Output word	the word address where you want the output bit status stored. For example, N7:0.

Sequential Function Chart Reset (SFR) (PLC-5/30, -5/40, -5/60 only)



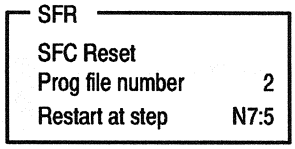
Description:
The SFR instruction resets the logic in a sequential function chart. When an SFR instruction goes true, the processor performs a postscan/lastscan on all active steps and actions in the selected file, and then resets the logic in the SFC on the next program scan. The chart remains in this reset state until the SFR instruction goes false. The SFR instruction also resets all retentive actions that are currently active.

Use the SFR instruction to handle situations for resetting your machines. For example, if a machine goes out of alignment, use the SFR instruction to reset the chart, align the machine (for example, with an SFC homing program) and then disable the SFR instruction to start the SFC again.

Entering Parameters

To program these instructions, you must provide the processor with the following information:

Example:



- **Program File Number** – a valid SFC program file number
- **Restart at Step** – enter one of the following:
  - a valid step reference number, 0 to 32767 (entering a 0 defaults to restarting at the initial step)
  - an integer address (that stores a step's reference number)
  - an address symbol (of an integer address that stores a step reference nuimber)

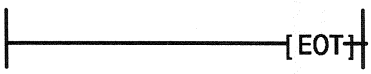
Important: The Restart at Step parameter is only available with PLC-5/30 series A, and PLC-5/40, -5/40L, -5/60, -5/60L series B processors. If you are using a PLC-5/40 or -5/60 series A processor, the SFC is automatically reset to the initial step.

A step number is a software-assigned reference number associated with each step. You must configure your SFC to display these numbers. For information on configuring your display, refer to the section "Configuring the SFC Display" in the Getting Started Using SFCs chapter.

Important: Make sure that the step is actually a step and not a transition or macro. This causes the processor to fault; the software does not perform a check. Also make sure that the step is not within a simultaneous branch.

End of Transition (EOT)

Example:



Description:
The EOT instruction should be last instruction in a transition file. If you do not place an EOT instruction in a transition file, the processor always evaluates the transition file as true.

User Interrupt Disable (UID) (PLC-5/30, -5/40, -5/60 only)



Description:
The UID instruction is used to temporarily disable interrupt programs, such as Selectable Timed Interrupts (STI) or Processor Input Interrupts (PII).

When the rung is true, the UID instruction increments an internal-interrupt-disable counter. As long as this counter value does not equal zero, the currently executing program **cannot** be interrupted by an STI or a PII. Also, if you have a subroutine call within a UIE/UID pair, that subroutine runs without interruption.

The UID instruction does not disable the user fault routine.

Important: Since the UID instruction makes a program un-interruptable, the processor's response time to an STI or PII event may be affected. The UID/UIE section of your program should be as short as possible. Leaving STIs and PIIs disabled for extended periods of time eventually leads to STI and PII overlap errors.

Important: Also note that if you are have a remote block transfer in an STI or PII and that remote block transfer instruction is within a UID/UIE section of your program, the main program scan stops until the block transfer completes.

User Interrupt Enable (UIE) (PLC-5/30, -5/40, -5/60 only)

—(UIE)—

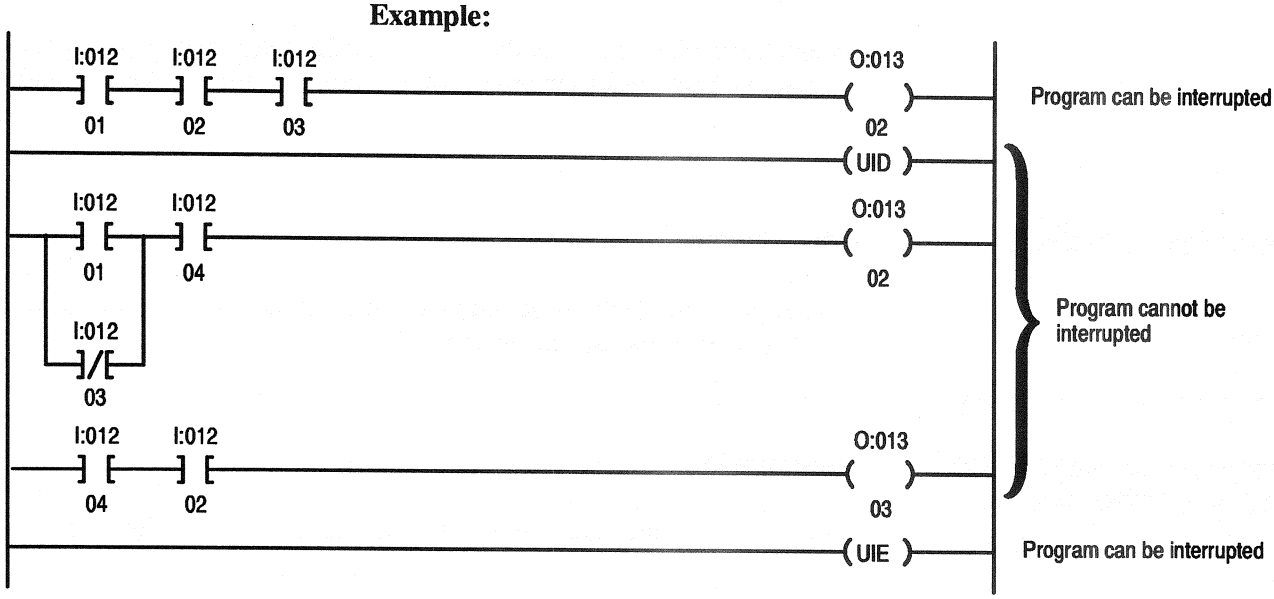
Description:

The UIE instruction is used to re-enable STI or PII interrupt programs.

When the rung is true and the internal interrupt-disable counter is greater than zero, the interrupt-disable counter is decremented.

When the counter equals zero, the program currently executing is then able to be interrupted again. If there are any interrupt programs pending, they are executed at this time.

Example:



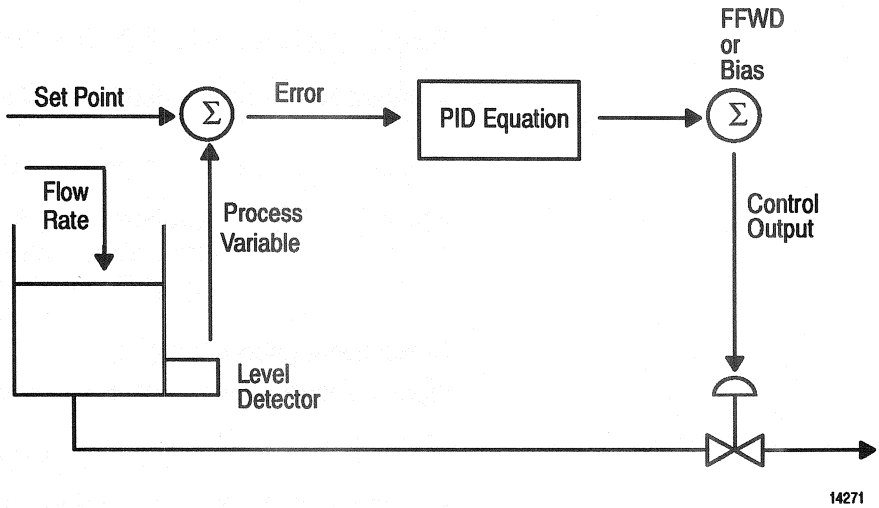
Chapter 14

Process Control Instructions PID

Using PID

PID closed loop control holds a process variable at a desired set point. Figure 14.1 shows a flow-rate/fluid level example.

Figure 14.1
PID Control Example



In the above example, the PID equation controls the process by sending an output signal to the control valve. The greater the error between setpoint and process variable input, the greater the output signal, and vice versa. An additional value (feedforward or bias) can be added to the control output as an offset. The goal of PID calculations is to maintain the process variable you are controlling at the set point.

For programming considerations, see the end of this chapter.

PID Features

The PID instruction lets the process monitor and control process loops for such quantities as pressure, temperature, flow rate, and fluid level. Features of the PID instruction include:

- PID equations expressed in ISA or Independent Gains
- input and output range from 0–4095 (12-bit analog)
- input scaling in engineering units
- zero-crossing dead band

- derivative term (can act on PV or error)
- direct or reverse acting control
- output alarms
- output limiting with anti-reset windup
- manual mode (with bumpless transfer)
- feedforward or output biasing
- displaying and monitoring PID values

Using PID Equations

The processor gives you four choices of PID algorithms:

- Standard equation with dependent gains:

  $$\text{Output} = Kc[(E) + 1/Ti \int_0^t (E)dt + Td*d(PV)/dt] + \text{Bias}$$

- Standard equation with derivative of error:

  $$\text{Output} = Kc[(E) + 1/Ti \int_0^t (E)dt + Td*d(E)/dt] + \text{Bias}$$

  where:
  - Kc = proportional gain
  - 1/Ti = reset gain in repeats per minutes
  - Td = rate gain in minutes

- Independent gains equation:

  $$\text{Output} = Kp(E) + Ki \int_0^t (E)dt + Kd*d(PV)/dt + \text{Bias}$$

- Independent gains equation with derivative of error:

  $$\text{Output} = Kp(E) + Ki \int_0^t (E)dt + Kd*d(E)/dt + \text{Bias}$$

  where:
  - Kp = proportional gain
  - Ki = integral gain in 1/seconds
  - Kd = derivative gain in seconds

Other variables used in any algorithm include:

Output	= Percentage of full scale
Bias	= Bias or Feedforward (0 - 4095)
E	= Error (scaled) SP-PV or PV-SP
PV	= Process Variable (scaled)

Comparison of Gain Constants

Compare standard and independent gains constants as follows:

Standard Gains Constants:	Independent Gains Constants:
Controller Gain Kc (dimensionless)	Proportional Gain Kp (dimensionless)
Reset Term 1/Ti (minutes per repeat)	Integral Gain Ki (inverse seconds)
Rate Term Td (minutes)	Derivative Term Kd (seconds)

Convert from standard to independent gains constants by substituting controller gain (Kc), reset (1/Ti), and rate (Td) values in the following formulas.

```
Kp = Kc unitless

Ki =   Kc    inverse seconds
     60 Ti

Kd = Kc(Td)60 seconds
```

Derivative Smoothing

The derivative term provides smoothing by means of a low pass filter. The filter's cutoff frequency is 16 times greater than the derivative term's corner frequency.

Setting Input/Output Ranges

The input module that measures the processor variable (PV) must have a full-scale binary range of 0–4095. The processor ignores the upper four most-significant bits of the 16-bit process variable.

The control output has the same range of 0–4095. You can set limits on the output to limit the output calculated by the PID instruction to any value in the range of 0 to 4095.

The tieback input (output tracking) from a manual control station must also have a range of 0–4095. The PID instruction uses the result to calculate the integral accumulated value, which allows for bumpless transfer from manual to automatic control.

The PID instruction also copies the tieback value into the control output storage location when in manual mode. Tieback input is only used when you use a hardware auto/manual station. Otherwise, set the tieback value to zero.

Implementing Scaling to Engineering Units

You can scale the setpoint and zero-crossing dead band values to engineering units. You can also display the process variable and error values in these same engineering units.

When you select scaling, the PID instruction scales the setpoint, dead band, process variable, and error values. You also have to do the following:

1. Enter the maximum and minimum values S_{max} and S_{min} in the PID control block (words 7 and 8). (See "Using the Control Block" below.) The S_{min} value corresponds to an analog value of zero for the lowest reading of the process variable; the S_{max} value corresponds to an analog value of 4095 for the highest reading of the process variable. These values represent process limits. Set S_{min} and S_{max} to 0 if you do not want scaling.

   For example, if you measure a scale of temperature from –73 (PV=0) to +1156 (PV=4095), enter –73 for S_{min} and 1156 for S_{max}.

   If the analog input module is not configured to return a value in the range 0–4095, see "Descaling Inputs" on page 14-23 in this chapter.

2. Reset bit 5 of word 0 in the PID control block (integer file type only). Set this bit only if you want to inhibit scaling the setpoint. You must inhibit setpoint scaling of a cascaded inner loop while scanning other loop variables.

3. Enter the setpoint,word 2, and dead band,word 9 (integer file type only), values in the same scaled engineering units. The control output (word 16) displays as a percentage of the 0–4095 range. The output the processor transfers to the output module is always unscaled.

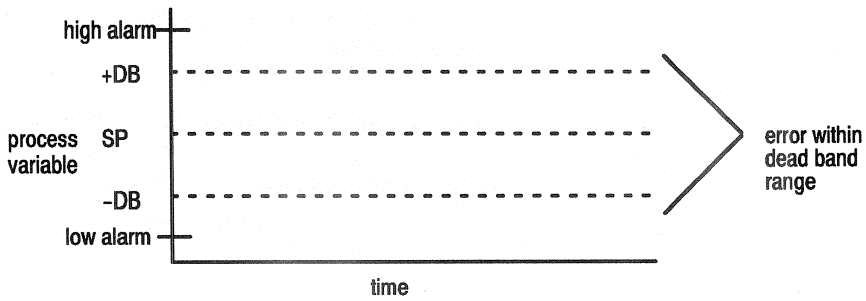


WARNING: Do not change scaling when the processor is in Run mode. The processor could fault and cause an undesirable process response, possible equipment damage, and personal injury.

Setting the Dead Band

The adjustable dead band lets you select an error range above and below the setpoint where output does not change as long as the error remains within this range.

This dead band lets you control how closely the process variable matches the setpoint without changing the output.



Using Zero-Crossing

Zero-crossing is dead band control that lets the instruction use the error for computational purposes as the process variable crosses into the dead band until the process variable crosses the setpoint. Once the process variable crosses the setpoint (error crosses zero and changes sign) and as long as the process variable remains in the dead band, the instruction considers the error value zero.

Enter your dead band value in word 9 of the control block (.DB word of a PD data file type). The dead band extends above and below the setpoint by the value you specify. Enter 0 to inhibit the dead band. If scaled, the dead band has the same scaled units as the setpoint.

Selecting the Derivative Term (Acts on PV or Error)

Derivative is a change of state variable. You can select whether the derivative term in either PID equation acts on changes in the processor variable or error value. Use bit 6 of word 0 in the control block (.DO word of a PD data file type)to select the type of derivative action you want.

Setting Output Alarms

You can set an output alarm on the control variable output at a selected value above or below the setpoint. When the instruction detects that the output has reached either value, the processor sets an alarm bit (bit 10 for lower limit, bit 9 for upper limit) in word 0 of the control block (.OLH and .OLL bits of a PD data file type). Alarm bits are reset by the instruction when the output comes back inside the limits. The instruction does not prevent the output from exceeding the the alarm values unless you select output limiting.

Enter the upper output alarm in word 11 (.MAXO) and the lower output alarm in word 12 (.MINO) of the control block. The processor handles output alarm values as a percentage of the output. If you do not want alarms, enter 0% for the lower alarm and 100% for the upper alarm.

Using Output Limiting

You can set an output limit (percent of output) on the control output. When the instruction detects that the output has reached a limit, it sets an alarm bit (bit 10 .OLL for the lower limit, bit 9 .OLH for the upper limit) in word 0 of the control block and prevents the output from exceeding either value. The instruction limits the output to 0 and 4095 if you do not specify a limit.

To use output limits, set the limit enable bit (bit 03 of word 0) and enter the upper limit in word 11 and the lower limit in word 12. Limit values are a percentage (0–100%) of the output.

Important: If you are using the PD data file type for the control block, the processor performs this function without you having to set bits.

Anit-reset Windup

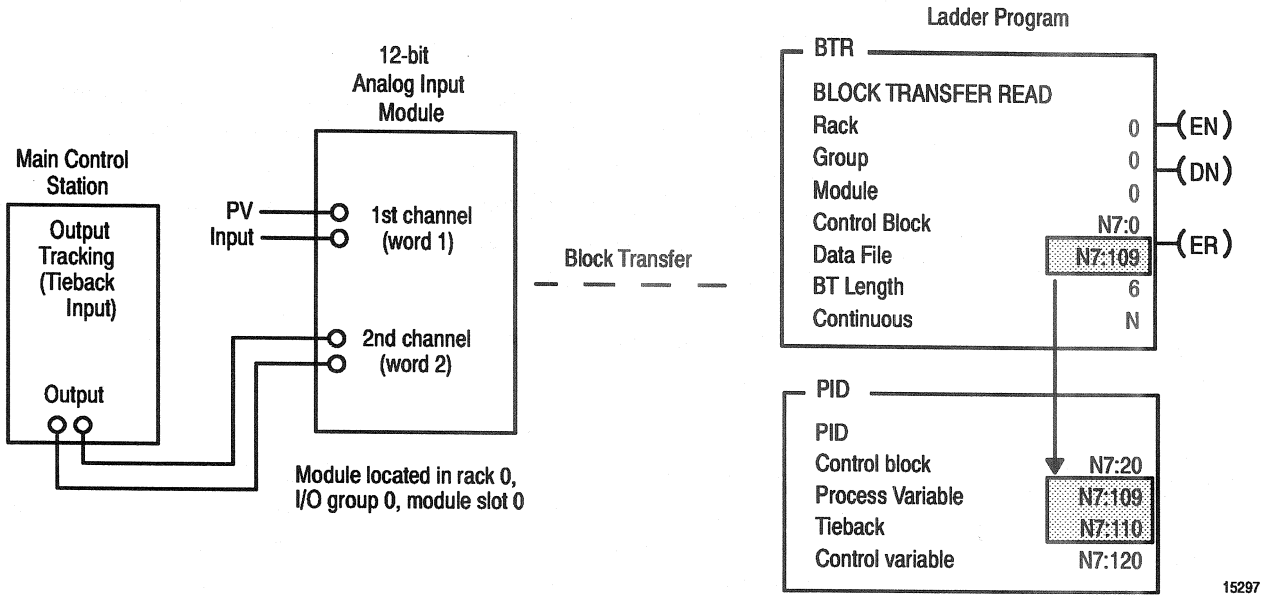
Anti-reset windup is a feature that prevents the integral term from becoming excessive when outputs reach a limit. When the sum of the PID and bias terms in the output reaches a limit, the instruction stops calculating the integral output term until the output comes back into range.

Using a Manual Mode Operation (with Bumpless Transfer)

Manual operation lets an output from a manual control station or from your ladder program override the calculated output of the PID instruction.

With a manual control station, you control the output device directly and override the PID instruction's output. You must feed the output value into the PID instruction's tieback input (Figure 14.2). The PID instruction uses this value to calculate the integral term value required to achieve a bumpless transfer when you switch from manual to automatic control.

Figure 14.2
Example Diagram for Moving Analog Inputs to a PID Instruction



Set Output

You can replace a manual control status with a thumbwheel and pushbutton switches, and simulate the PID function with ladder logic. Your program should

- set the set output bit (bit 04, word 0 PD.SWM) when you enter the PID instruction or from a switch input to the ladder program.
- move the desired percentage of output from elsewhere in the program into the set output word (word 10 PD.SO) in the control block. The PID instruction sets its output to this percentage value.

If the set output value is greater than the upper CV limit or lower than the lower CV limit and output limiting is enabled and the PID instruction is in set output mode, the processor uses the actual output (not the set output value) to calculate the integral accumlator term for calculating bumpless transfer.

Feedforward or Output Biasing

You can feedforward a disturbance from the system or bias output by feeding either of these values into the PID instruction's feedforward/bias word (word 6 PD.BIAS) of the control block. Either value must have a range of 0–4095.

The feedforward value represents a disturbance fed into the PID instruction before the disturbance has a chance to change the process variable. Feedforward is often used to control processes with a transportation lag. For example, a feedforward value representing "cold water poured into a warm mix" could boost the output faster than waiting for the process variable to change as a result of mixing.

A bias value can be used to compensate for a steady-state loss of energy from the controlled process.

Resume Last State

With the resume-last-state function, you can make full use of the analog output module's hold-last-state function. The resume-last-state function lets the PID instruction resume calculating the integral term of the PID algorithm from its last output value (instead of zero) when returning to RUN mode.

If you are using an integer data file for the control block, set the bits according to the guidelines below. If you are using a PD data file type for the control block, the processor saves the integral accumulator and uses it when going from Program to Run mode.

Use this function as follows:

- Set word 0, bit 7 if you configured the analog output module to hold last state if a fault occurs and when changing from RUN to PROGram mode
- Reset word 0, bit 7 if you configured the analog output module to turn off if a fault occurs and when changing from RUN to PROGram mode

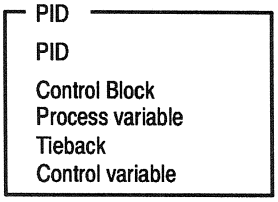


WARNING: If you want to use this function, set bit 7 only after the PID instruction has executed at least once (at start-up) or when returning to RUN mode. If you do not let the PID instruction execute at least once, unpredictable machine operation may occur causing possible damage to equipment and/or injury to personnel.

This function is available with the following processors:

PLC-5/12 Series A/Rev C and later	PLC-5/40 Series A and later
PLC-5/15 Series B/Rev H and later	PLC-5/60 Series A and later
PLC-5/25 Series A/Rev D and later	PLC-5/40L Series A and later
PLC-5/30 Series A and later	PLC-5/60L Series A and later

PID Instruction



Description:
The PID instruction is an output instruction that controls physical properties, such as temperature, pressure, liquid level, or flow rate of process loops.

The PID instruction controls a PID loop with inputs from an analog input module and an output to an analog output module. For temperature control, you can convert the analog output to a time proportioning on/off output for driving a heater or cooling unit.

You must execute this instruction periodically at constant intervals using a timer, a selectable timed interrupt (STI), or a real-time sampling. The ladder program can interact with the PID algorithm by changing variables during operation, or you can change variables from a programming terminal or from stations on a communications link such as Data Highway or Data Highway Plus.

Using Status Bits

To processor uses the following status bits:

- **Enable Bit .EN** (bit 15 PD.EN) is set when the rung makes a false-to-true rung transition to indicate the instruction is enabled.
- **Done Bit .DN** (bit 13) is set after the instruction finishes operating on the PID equations. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition. If you are using a PD data file for the control block, the processor does not use this status bit.

Entering Parameters

When you enter the instruction, you must specify four addresses that are fundamental to the operation of the instruction. After you enter these addresses, the programming software displays a screen from which you enter the operating parameters of the instruction.

The addresses that you enter are:

- **Control Block** is a file that stores PID status and control bits, constants, variables, and internally used parameters.

  Based on the data type you use, a different configuration screen appears for you to enter PID information (see the next sections for more information):

  - If you have a PLC-5/30, -5/40 or -5/60 processor, you can use an integer file or a PID data file (PD). Using a PID file, words 0, 1 are the status words; words 2–80 store the PID values.

- If you have a PLC-5/10, -5/12, -5/15, -5/25, you must use an integer file (N) for your control block. Using an integer file, word 0 is the status word; words 1–22 store the PID values.

- **Process Variable** is a word address that stores the process input value.
- **Tieback** is a word address used to implement bumpless transfer when using a manual control station. The tieback is an output of a BTR instruction from the station.
- **Output** is a word address to which the PID instruction sends its calculated PID output value.

Using an Integer Data File Type for the Control Block

After you enter the above addresses (using an N control block) and accept the rung that contains the PID instruction, the software displays the data monitor screen for the PID instruction (Figure 14.3).

Figure 14.3
Data Monitor Screen for PID (Integer Control Block)

```
            equation: 1 (0:AB/1:ISA)              feed forward:      7
                mode: 0 (0:auto/1:manual)    max scaled input:    300
               error: 1 (0:SP-PV/1:PV-SP)    min scaled input:      0
      output Limiting: 1 (0:NO/1:YES)              deadband:       15
      set output mode: 0 (0:NO/1:YES)       set output value %:     0
     setpoint scaling: 0 (0:YES/1:NO)         upper CV limit %:    100
     derivative input: 1 (0:PV/1:error)       lower CV limit %:      0
    last state resume: 0 (0:NO/1:YES)          scaled PV value:     33
      deadband status: 1                          scaled error:      8
  upper CV Limit alarm: 0                         current CV %:      7
  lower CV Limit alarm: 0
 setpoint out of range: 0
             PID done: 0                                setpoint:    25
          PID enabled: 0      proportional gain (Kc)      [.01]:    10
                              res. time (Ti) [.01 mins/repeat]:     1
                              derivative rate (Td)  [.01 mins]:     2
                              loop update time     [.01 secs]:     10


Enter value or press <ESCAPE> to exit monitor.
N10:0/0 =
Program    Forces:None      Data:Decimal                PLC-5/15 File DRILL1
```

Parameter Descriptions:

The data monitor screen for the PID instruction shows the following information, some of which is display only; some of which you specify the values (Table 14.A).

Table 14.A
PID Parameter Descriptions (Integer Control Block)

Parameter	Description:
Equation	Enter whether you want to use independent (0) or dependent (1) gains. Displays one of the following: INDEPENDENT (0) - for independent gains DEPENDENT (1) - for dependent gains (ISA) Use dependent gains when you want to use standard loop tuning methods. Use independent gains when you want the three gain constants (P, I, and D) to operate independently.
Mode	Displays operating mode: AUTO (0) - automatic PID control MANUAL (1) - control from a manual control station Sets the use of the tieback parameter for manual operation
Error	Displays one of the following error value: Direct acting: 0 = SP-PV Reverse acting: 1 = PV-SP
Output Limiting	Displays whether or not the instruction clamps the output at the high and low limiting values. Displays one of the following: NO (0) - output not clamped YES (1) - output clamped The PID algorithm has an anti-reset windup feature that prevents the integral term from becoming too large when the output reaches the high or low alarm limits. If the limits are reached, the algorithm stops calculating the integral term until the output comes back into range.
Set output mode	Selects the use of set output value % for manual operation
Setpoint scaling	Selects if the setpoint is to be interpreted as a value in the engineering units of an unscaled (0 to 4095) value
Derivative input	Selects if derivative term is based on changes in PV or on changes in error
Deadband status	Set if the PV is inside the selected deadband range; reset if it is not
Upper CV limit alarm	Set if caculated CV is greated than the CV upper limit word %
Lower CV limit alarm	Set if calculated CV is less than the CV lower limit word %
Setpoint out of range	Displays whether or not the setpoint is out of the range of engineering units you selected on the PID Configuration screen. Displays one of the following: NO (0) - SP within range YES (1) - SP out of range **Important:** A major processor fault occurs if the SP is out of range when the instruction is first enabled.
PID done	Displays whether the PID instruction has completed.
PID enabled	Displays whether the the PID instruction is enabled.
Feed forward Bias	Enter a value from 0 to ±4095 for the amount of feed forward. The ladder program can enter a feedforward value to bump the output in anticipation of a disturbance. This value is often used to control a process with transportation lag.

Parameter	Description:
Max scaled input	Enter the integer number (-32,768 - 32,767) that is the maximum value available from the analog module. For example, use 4095 for a module whose range is 0-4095.
Min scaled input	Enter the number that is the minimum value available from the analog module. For example, use 0 for a module whose range is 0-4095.
Dead band	For an unscaled deadband, enter a value in the engineering units you selected on the PID Configuration screen. Valid range is 0 to 4095 unscaled, -32, 768 to +32, 767 scaled. **Important:** The deadband is zero crossing.
Set output value %	Enter a percent (0-100%) to use as the CV Output when 'set output mode' is selected.
Upper CV limit %	Enter a percent (0-100) above which the algorithm clamps the output.
Lower CV limit %	Enter a percent (0-100) below which the algorithm clamps the output.
Scaled PV value	Displays data from the analog input module that the instruction scales to the same engineering units that you selected for the setpoint.
Scaled error	Displays the current error in scaled engineering units
Current CV %	Displays current controlled variable output valve as a percent
Setpoint	Enter an integer value. Valid range is 0-4095 (unscaled) or S_{min} - S_{max} (scaled engineering units).
Proportional gain (Kc)	Enter an integer. Valid range is 0-327.67 (unitless) or K_p 0 to 327.67
Reset time (Ti) minutes/repeat	Enter an integer. Valid range is 0-327.67 (unitless) or K_I 0 to 32.76
Derivative rate (Td)	Enter an integer. Valid range is 0-327.76 or K_D 0 to 327.67
Loop update time	Enter an update time (greater than or equal to 0.01 seconds) at five to ten times the natural period of the load (load time constant). Valid range is 0.01 - 327.67 seconds. The load time constant should be greater than: 1ms(algorithm) + block transfer time (ms) Periodically enable the PID instruction at a constant interval equal to the update time. For update times of less than 100 msec, use an STI. When update times of greater than 100 msec, use a timer or a real-time sampling. **CAUTION:** If you omit an update time or enter a negative update time, a major fault occurs the first time the processor runs the PID instruction.

Using Control Block Values

Word 0 of the control block contains status and control bits.Table 14.B shows the values stored in each word of the control block.

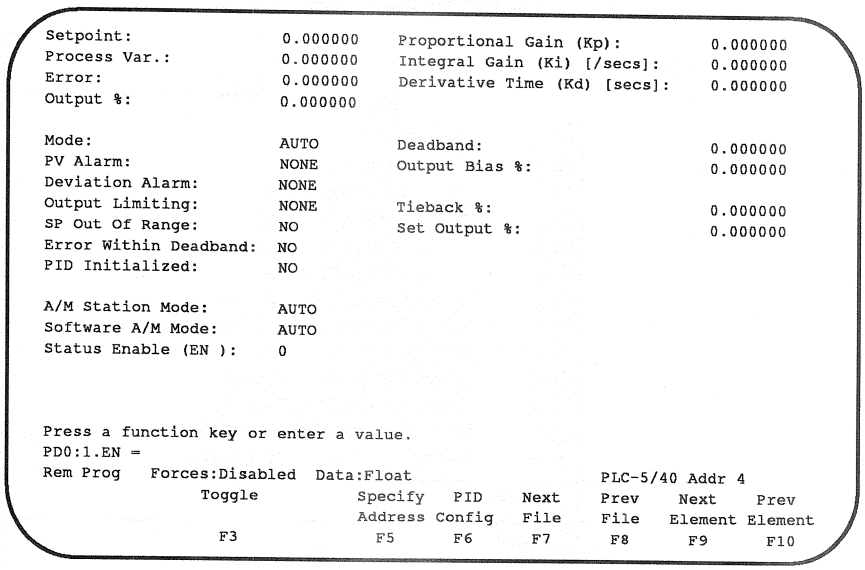
Table 14.B
PID Control Block (Integer Control Block)

Word:	Contains:		Term:	Range:
0	Bit 15 Enabled (EN) Bit 13 Done (DN) Bit 11 Set point out of range Bit 10 Output alarm, lower limit Bit 9 Output alarm, upper limit Bit 8 DB, set when error is in DB Bit 7 Resume last state (0=yes; 1=hold last state) Bit 6 Derivative action (0=PV, 1=error) Bit 5 Setpoint descaling (0=no, 1=yes) Bit 4 Set output (0=no, 1=yes) Bit 3 Output limiting (0=no, 1=yes) Bit 2 Control (0=direct, 1=reverse) Bit 1 Mode (0=automatic, 1=manual) Bit 0 Equation (0=independent, 1=ISA)			
1	reserved			
2	Setpoint		SP	0 -4095 (unscaled) S_{min} - S_{max} (scaled)
3	**Independent:**	Proportional gain (unitless)	K_p	0 - 327.67
	ISA:	Controller gain (unitless)	K_C	0 - 327.67
4	**Independent:**	Integral gain (1/sec)	K_I	0 - 32.767
	ISA:	Reset term (minutes per repeat)	T_I	0 - 327.67
5	**Independent:**	Derivative gain	K_D	0 - 327.67
	ISA:	Rate term x100 (minutes)	T_D	0 - 327.67
6	Feedforward or bias		FF/Bias	0 - 4095
7	Maximum scaling		S_{max}	$^{-}$32,768 - $^{+}$32,767
8	Minimum scaling		S_{min}	$^{-}$32,768 - $^{+}$32,767
9	Dead band		DB	0 - 4095 (unscaled) S_{min} - S_{max} (scaled)
10	Set output		SETOUT	0 - 100%
11	Maximum output limit (% of output)		L_{max}	0 - 100%
12	Minimum output limit (% of output)		L_{min}	0 - 100%
13	Loop update time (seconds)		dt	0.01 - 327.67
14	Scaled PV value (displayed)			S_{min} - S_{max}
15	Scaled error value (displayed)			S_{min} - S_{max}
16	Output (% of 4095)		CV	0 - 100%
17 - 22	internal storage; do not use			

Using a PD File Type for the Control Block

After you enter the above addresses (using a PD control block) and accept the rung that contains the PID instruction, the software displays the data monitor screen for the PID instruction (Figure 14.3).

Figure 14.4
Data Monitor Screen for PID (PD Control Block)



Parameter Descriptions:

The data monitor screen for the PID instruction shows the following information, some of which is display only; some of which you specify the values (Table 14.C).

Table 14.C
PID Parameter Descriptions (PD Control Block)

Parameter	Address Mnemonic:	Description:
Setpoint	.SP	Enter a floating-point number in the same engineering units that are on the PID Configuration screen. Valid range is $-3.4\ E^{+38}$ to $+3.4\ E^{+38}$.
Process Variable	.PV	Displays data from the analog input module that the instruction scales to the same engineering units that you selected for the setpoint.
Error	.ERR	Displays one of the following error values: Direct acting: Error = SP-PV Reverse acting: Error = PV-SP
Output %	.OUT	Displays the PID algorithm control output value (0–100%).

Parameter	Address Mnemonic:	Description:
Mode	.MO .MO=0 .MO=1 .SWM=1	Displays operating mode: AUTO - automatic PID control MANUAL - control from a manual control station SW MANUAL - simulated manual control from the data monitor or ladder program
PV Alarm	 .PVHA=1 .PVLA=1	Displays whether the PV is within or exceeds the high or low alarm limits you selected on the PID Configuration screen. Displays one of the following: NONE- PV within alarm limits HIGH - PV exceeds high alarm limit (used with deadband) LOW - PV exceeds low alarm limit (used with deadband)
Deviation Alarm	 .DVPA=1 .DVNA=1	Displays whether the error is within or exceeds the high or low deviation alarms you selected on the PID Configuration screen. Displays one of the following: NONE - error within deviation alarm limits POSITIVE - error exceeds high alarm (used with deadband) NEGATIVE - error exceeds low alarm (used with deadband)
Output Limiting	 .OLH=1 .OLL=1	Displays whether or not the instruction clamps the output at the high and low limiting values (.MAXO and .MINO) you selected on the PID Configuration screen. Displays one of the following: NONE - output not clamped HIGH - output clamped at the high end (.MAXO) LOW - output clamped at the low end (.MINO) The PID algorithm has a anti-reset-windup feature that prevents the integral term from becoming too large when the output reaches the high or low alarm limits. If the limits are reached, the algorithm stops calculating the integral term until the output comes back into range.
SP Out of Range	 .SPOR=0 .SPOR=1	Displays whether or not the setpoint is out of the range of engineering units you selected on the PID Configuration screen. Displays one of the following: NO - SP within range YES - SP out of range **Important:** A major processor fault occurs if the SP is out of range when the instruction is first enabled.
Error Within DB	 .EWD=0 .EWD=1	Displays whether the error is within or exceeds the deadband value you enter on this screen. The deadband is zero crossing. Displays one of the following: RESET - Error exits the deadband zone SET - Error crosses the deadband centerline
PID Initialized	 .INI=0 .INI=1	Each time you change a value in the control block, the PID instruction takes over twice as long to execute (until initialized) on the first scan. Displays one of the following: NO - PID instruction not initialized after you changed control block values YES- PID instruction remains initialized because you did not change any control block values **WARNING:** Do not change the range of input or engineering units when running. If you must do this, then you must reset this bit to re-initialize. Otherwise, the instruction will malfunction with possible damage to equipment and injury to personnel.

Parameter	Address Mnemonic:	Description:
A/M Station Mode	.MO=0 .MO=1	Enter whether you want automatic (0) or manual (1) PID control. Displays one of the following: AUTO (0) - automatic PID control MANUAL (1) - manual PID control Manual control specified that an output from a manual control station overrides the calculated output of the PID algorithm. **Important:** Manual overrides Set Output Mode.
Software A/M Mode	.SWM=0 .SWM=1	Enter whether you want automatic PID (0) control or Set Output Mode (1), for software-simulated control. Displays one of the following: AUTO (0) - automatic PID control SW MANUAL (1) - software-simulated PID control You can simulate a manual control station with the data monitor when you program a single loop. To do this, set .SWM to SW MANUAL and enter a Set Output Percent value. You can simulate a manual control station with ladder logic, pushwheels, and pushbutton switches when you program several loops. To do this, set .SWM to SW MANUAL in your and move a value into the set output element .SO.
Status Enable	.EN=0 .EN=1	Enter whether to use (1) or inhibit (0) this bit which displays the rung condition so you can see whether the PID instruction is operating. Displays one of the following: 0 - instruction not executing 1 - instruction executing
Proportional Gain	.KP	Enter a floating-point value. Valid range for independent or standard gains is 0 to 3.4 E^{+38} (unitless)
Integral Gain	.KI	Enter a floating-point value. Valid range for independent gains is 0 to 3.4 E^{+38} inverse seconds; valid range for standard gains is 0 to 3.4 E^{+38} repeats per minute
Derivative Gain	.KD	Enter a floating-point value. Valid range for independent gains is 0 to 3.4 E^{+38} seconds; valid range for standard gains is 0 to 3.4 E^{+38} minutes.
Deadband	.DB	Enter a floating-point value in the engineering units you selected on the PID Configuration screen. Valid range is 0 to 3.4 E^{+38}. See Error Within DB value (.EWD).
Output Bias %	.BIAS	Enter a value (-100 to +100) to represent the percentage of output you want to feed forward or use as a bias to the output. The bias value can compensate for steady-loss of energy from the system. The ladder program can enter a feedforward value to bump the output in anticipation of a disturbance. This value is often used to control a process with transportation lag.

Parameter	Address Mnemonic:	Description:
Tieback %	.TIE	Displays a number (0 to 100) representing the percent of raw tieback (0-4095) from the manual control station. The PID algorithm uses this number to achieve bumpless transfer when switching from manual to auto mode.
Set Output %	.SO	Enter a percent (0 to 100), from this screen or a ladder program, to represent the software-manually controlled output. When you select software-simulated control (.SWM=1), the PID instruction overrides the algorithm with the set output value (0 to 4095) for transfer to the output module, and copies it to .OUT for display as a percent. The transfer to software-simulated control is bumpless because .SO (under your control) starts with the last automatic algorithm output. Do not vary .SO until after the transfer. To achieve bumpless transfer when changing from software-simulated control to automatic control, the PID algorithm changes the integral term so the the output is equal to the set output value.

The data monitor screen for the PID instruction provides access to the PID Configuration screen. From this screen (Figure 14.5), you can define characteristics of the PID instruction.

Figure 14.5
PID Configuration Screen (PD Control Block)

```
PID Equation:           INDEPENDENT Engineering Unit Maximum:       0.000000
Derivative Of:          PV          Engineering Unit Minimum:       0.000000
Control Action          DIRECT
PV Tracking:            NO          Input Range Maximum:            0.000000
                                    Input Range Minimum:            0.000000

Update Time (secs):     0.000000    Output Limit High %:            0.000000
                                    Output Limit Low %:             0.000000

Cascaded Loop:          NO          PV Alarm High:                  0.000000
Cascaded Type:          -           PV Alarm Low:                   0.000000
Master To This Slave:   -           PV Alarm Deadband:              0.000000

                                    (+) Deviation Alarm:            0.000000
                                    (-) Deviation Alarm:            0.000000
                                    Deviation Alarm Deadband:       0.000000



Press a function key or enter a value.
PD0:1.PE =
Rem Prog   Forces:Disabled  Data:Float      Addr:Decimal PLC-5/40 Addr 4
              Toggle          Specify   PID    Next    Prev    Next    Prev
                              Address Monitor  File    File  Element Element
                F3              F5      F6     F7      F8      F9      F10
```

Table 14.D describes the PID characteristics you can configure on the PID Configuration screen.

Table 14.D
PID Configuration Characteristics (PD Control Block)

Parameter:	Address Mnemonic:	Description:
PID Equation	.PE=0 .PE=1	Enter whether you want to use independent (0) or dependent (1) gains. Displays one of the following: INDEPENDENT (0) - for independent gains DEPENDENT (1) - for dependent gains Use dependent gains when you want to use standard loop tuning methods. Use independent gains when you want the three gain constants (P, I, and D) to operate independently.
Derivative of	.DO=0 .DO=1	Enter whether you want the derivative of the PV (0) or the error (1). Displays one of the following: PV (0) - for PV derivative ERROR (1) - for error derivative Select the PV derivative for more stable control when you do not change the setpoint often. Select the error derivative for fast responses to setpoint changes when the algorithm can tolerate overshoots.
Control Action	.CA=0 .CA=1	Enter whether you want direct (0) or reverse acting (1). Displays one of the following: DIRECT (0) - for direct acting REVERSE (1) - for reverse acting Select reverse acting when driving the PV to a smaller SP as in a cooling application. Select direct acting when driving the PV to a larger SP as in a heating application.
PV Tracking	.PVT=0 .PVT=1	Enter whether you do not (0) or do (1) want PV tracking. Displays one of the following: NO (0) - for no tracking YES (1) - for PV tracking Select no tracking if the algorithm can tolerate a bump when switching from manual to automatic control. Select PV tracking if you want the setpoint to track the PV in manual control for bumpless transfer to automatic control.
Update Time	.UPD	Enter an update time (greater than or equal to .01 seconds) at 1/5 to 1/10 the natural period of the load (load time constant). The load time constant should be greater than: 3ms(algorithm) + block transfer time (ms) Periodically enable the PID instruction at a constant interval equal to the update time. When the program scan time is close to the required update time, use an STI to ensure a constant update interval. When the program scan is several times larger than the required update time, use a timer. **CAUTION:** If you omit an update time or enter a negative update time, a major fault occurs the first time the processor runs the PID instruction.
Cascade Loop	.CL=0 .CL=1	Enter whether this loop is not (0) or is (1) used in a cascade of loops. Displays one of the following: NO (0) - not used in a cascade YES (1) - used in a cascade

Parameter:	Address Mnemonic:	Description:
Cascade Type	.CT=0 .CT=1	If this loop is part of a cascade of loops, enter whether this loop is the master (1) or a slave (0). Displays one of the following: SLAVE (0) – for a slave loop MASTER (1) – for a master loop
Master to This Slave	.ADDR	If this loop is a slave loop in a cascade, enter the control block address of the master. Tieback is ignored in the master loop of a cascade. When you change cascaded loops to manual control, the slave forces the master into manual control. When PV tracking is enabled, the order of events is: Slave.SP > Master.TIE > Master.OUT > Slave.SP When you return to automatic control, change the slave first, then the master.
Engineering Unit Max	.MAXS	Enter the floating-point value in engineering units that corresponds to the analog module's full scale output. Valid range is $-3.4\ E^{+38}$ to $+3.4\ E^{+38}$. **CAUTION:** Do not change this value during operation because a processor fault might occur.
Engineering Unit Min	.MINS	Enter the floating-point value in engineering units that corresponds to the analog module's zero output. Valid range is $-3.4\ E^{+38}$ to $+3.4\ E^{+38}$. **CAUTION:** Do not change this value during operation because a processor fault might occur.
Input Range Max	.MAXI	Enter the floating-point number ($-3.4\ E^{+38}$ to $+3.4\ E^{+38}$) that is the maximum value available from the analog module. For example, use 4095 for a module whose range is 0 to 4095.
Input Range Min	.MINI	Enter the floating-point number ($-3.4\ E^{+38}$ to $+3.4\ E^{+38}$) that is the minimum value available from the analog module. For example, use 0 for a module whose range is 0 to 4095.
Output Limit High %	.MAXO	Enter a percent (0 to 100) above which the algorithm clamps the output.
Output Limit Low %	.MINO	Enter a percent (0 to 100) below which the algorithm clamps the output.
PV Alarm High	.PVH	Enter a floating-point number ($-3.4\ E^{+38}$ to $+3.4\ E^{+38}$) that represents the highest PV value that the system can tolerate.
PV Alarm Low	.PVL	Enter a floating-point number ($-3.4\ E^{+38}$ to $+3.4\ E^{+38}$) that represents the lowest PV value that the system can tolerate.
PV Alarm Deadband	.PVDB	Enter a floating-point number (0 to $3.4\ E^{+38}$) that is sufficient to minimize nuisance alarms. This is a one-sided deadband. The alarm bit (.PVH or .PVL) is not set until the PV crosses the deadband and reaches the alarm limit (DB zero point). The alarm bit remains set until the PV passes back through and exits from the deadband.
Deviation Alarm (+)	.DVP	Enter a floating-point number (0 to $3.4\ E^{+38}$) that specifies the greatest error deviation above the setpoint that the system can tolerate.

Parameter:	Address Mnemonic:	Description:
Deviation Alarm (-)	.DVN	Enter a floating-point number (-3.4 E^{+38} to 0) that specifies the greatest error deviation below the setpoint that the system can tolerate.
Deviation Alarm Deadband	.DVDB	Enter a floating-point number (0 to 3.4 E^{+38}) that is sufficient to minimize nuisance alarms. This is a one-sided deadband. The alarm bit (.DVP or .DVN) is not set until the error crosses the deadband and reaches the alarm limit (DB zero point). The alarm bit remains set until the error passes back through and exits from the deadband.

Using Control Block Values

Words 0 and 1 of the control block contains status and control bits. Table 14.E shows the values stored in each word of the control block.

Table 14.E
PID Control Block

Word:	Contains:		Range:
0	**Control/Status Bits** Bit 15 Enabled (EN) Bit 9 Cascade Selection (master, slave) Bit 8 Cascade Loop (0=no, 1=yes) Bit 7 Process variable tracking (0=no, 1=yes) Bit 6 Derivative action (0=PV, 1=error) Bit 4 Set output (0=no, 1=yes) Bit 2 Control (0=direct, 1=reverse) Bit 1 Mode (0=automatic, 1=manual) Bit 0 Equation (0=independent, 1=ISA)		
1	**Status Bits** Bit 12 PID initialized (0=no, 1=yes) Bit 11 Set point out of range Bit 10 Output alarm, lower limit Bit 9 Output alarm, upper limit Bit 8 DB, set when error is in DB Bit 3 Error is alarmed low Bit 2 Error is alarmed high Bit 1 Process variable (PV) is alarmed low Bit 0 Process variable (PV) is alarmed high		
2, 3	Setpoint		-3.4 E^{+38} to +3.4 E^{+38}
4, 5	**Independent:**	Proportional gain (unitless)	0 to +3.4 E^{+38}
	ISA:	Controller gain (unitless)	0 to +3.4 E^{+38}
6, 7	**Independent:**	Integral gain (1/sec)	0 to +3.4 E^{+38}
	ISA:	Reset term (minutes per repeat)	0 to +3.4 E^{+38}
8, 9	**Independent:**	Derivative gain	0 to +3.4 E^{+38}
	ISA:	Rate term x100 (minutes)	0 to +3.4 E^{+38}
10, 11	Feedforward or bias		-3.4 E^{+38} to +3.4 E^{+38}
12, 13	Maximum scaling		-3.4 E^{+38} to +3.4 E^{+38}
14, 15	Minimum scaling		-3.4 E^{+38} to +3.4 E^{+38}

Word:	Contains:	Range:
16, 17	Dead band	0 to +3.4 E^{+38}
18, 19	Set output	0 - 100%
20, 21	Maximum output limit (% of output)	0 - 100%
22, 23	Minimum output limit (% of output)	0 - 100%
24, 25	Loop update time (seconds)	
26, 27	Scaled PV value (displayed)	
28, 29	Scaled error value (displayed)	
30, 31	Output (% of 4095)	0 - 100%
32, 33	Process variable high alarm value	-3.4 E^{+38} to +3.4 E^{+38}
34, 35	Process variable low alarm value	-3.4 E^{+38} to +3.4 E^{+38}
36, 37	Error high alarm value	0 to +3.4 E^{+38}
38, 39	Error low alarm value	-3.4 E^{+38} to 0
40, 41	Process variable alarm deadband	0 to +3.4 E^{+38}
42, 43	Error alarm deadband	0 to +3.4 E^{+38}
44, 45	Maximum input value	-3.4 E^{+38} to +3.4 E^{+38}
46, 47	Minimum input value	-3.4 E^{+38} to +3.4 E^{+38}
48, 49	Tieback value for manual control (0-4095)	0 - 100%
51	Master PID file number	0 - 999
52	Master PID element number	0 - 999
54-80	internal storage; do not use	

Programming Considerations

When you program a PID instruction, do not change the following values when the processor is in Run mode:

- choice of ISA or independent gains equation because the PID gains constants are not directly interchangeable
- scaling values S_{min} and S_{max} because a change could place the setpoint out of range and could change the dead-band range
- choice of derivative action based on change in PV or change in error because internal values will change

Run Time Errors

If the processor encounters one of the following conditions, the processor produces a run time error when it tries to run the instruction:

- loop update time is zero or negative
- setpoint (SP) is out of range (SP < S_{min} or SP > S_{max})

  If you change SP, S_{min}, or S_{max} the PID instruction first tries to use the previously valid setpoint, continues PID control, and sets the setpoint out of range error bit. If the instruction finds no previously valid setpoint, it produces a run time error.

If you enter negative values for K_p, K_I, K_D, K_C, T_I, or T_D, the PID instruction substitutes zero for the negative value. This inhibits that term in the equation without producing a run time error.

Transferring Data to the PID Instruction

Use block transfer instructions to transfer data between analog I/O modules and the PID instruction. Use a BTR instruction for input values (process variable and tieback); use a BTW instruction for the control output.

Make each block transfer file address (data file entry) the same address in the PID for the process variable, tieback, and control output, respectively.

Not all Allen-Bradley analog input modules enter data in the same format. You must determine where to store channel data. For example, temperature sensing modules (such as the 1771–IR and 1771–IXE) place status words in front of the words that contain channel data. For information about where an analog module stores channel data, see the documentation for the module.

Loop Considerations

The number of PID loops, loop update time, and location of 12-bit analog input modules are important considerations for using the PID instruction

Number of PID Loops

The number of PID loops that the processor can handle depends on the update time required by the loops. The longer the update time and the less sophisticated the loop control, the more loops the processor can control.

The sum of the worst-case block transfer time associated with the analog inputs plus the time required for 1 program scan should be less than the update time required by the loops.

Loop Update Time

The PID instruction calculates a new control output whenever its rung changes from false to true when using an integer data file for the control block. For the instruction to operate predictably, the update time must be equal to the rate at which the PID rung changes between false and true. Deviation in toggle rate from the update time substantially degrades the accuracy of PID calculations.

You should program fast response loops (update times less than 100 msec) in the selectable timed interrupt (STI) along with the corresponding block transfer instructions. Unlatch the PID enable bit to force execution every STI scan (if you are using a PD data file for the control block you do not have to unlatch the enable bit). You must place corresponding analog I/O modules in the local chassis when you see this configuration.

Program slower response loops (update times of greater than 100 msec) in the main ladder program and use timers or real-time sampling to control the update time.

Descaling Inputs

The PID instruction must use unscaled (0–4095) data from analog input modules. The analog input modules you can use may have either scaled or unscaled ranges. When possible, select the unscaled range of 0–4095.

However, some modules such as the 1771-IR and 1771-IXE temperature sensing modules, cannot generate data in an unscaled range. For these modules, you must program arithmetic logic to convert the scaled output to the unscaled range for the PID instruction. If you are using a PD data file for the control block, the processor performs this descaling internally (see the descriptions of .MAXI and .MINI in the PID configuration characteristics, page 14-18).

Use this equation to to convert scaled outputs:

$$M_2 = (M_1 - S_{min1}) \frac{4095}{(S_{max1} - S_{min1})}$$

Variable	Description
M_2	calculated output
M_1	measured value from the module in scaled units
S_{max1}	scaled maximum value from the module
S_{min1}	scaled minimum value from the module
$S_{max1} - S_{min1}$	scaled range from the module

For example, the reading from a 1771-IXE module for type J thermocouple is 170°. To convert this to an unscaled value, use these values:

$$M_2 = [170 - (-200)] \frac{4095}{[1200 - (-200)]}$$

$M_2 = 1082$ unscaled

If you are sure that the temperature of your process will always remain within a specific range, you can set the limits for S_{min1} and S_{max1} instead of the minimum and maximum values for the thermocouple module. This technique improves the resolution of the process variable.

WARNING: If you set the limits in place of the lower and upper temperature limits of the thermocouple or RTD module, you must keep the process within the limits you specify. Failure to keep the process within the limits could cause unpredictable operation, damage to equipment, or injury to personnel.

Figure 14.6 shows the ladder logic you need to add to your PID program. Table 14.F lists the variables in this example.

Figure 14.6
Descaling PID Values Example

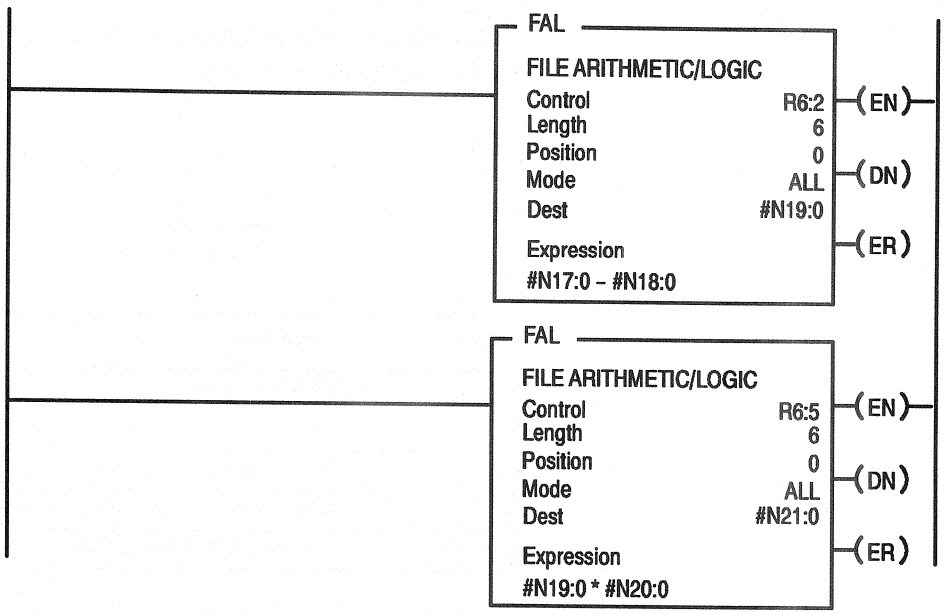


Table 14.F
Variables for the Descaling PID Values Example

Variable	Description
S_{max}	maximum scaling value
S_{min}	minimum scaling value
$K = \frac{4095}{S_{max} - S_{min}}$	constant for each channel
#N17:0	contains M_1 values for each channel
#N18:0	contains S_{min} constants for each channel
#N19:0	contains the result of $M_1 - S_{min}$ for each channel
#N20:0	location where you store K for each channel
#N21:0	contains the resulting unscaled value for each channel

PID Examples

The following examples assume that the channel data is stored starting at the beginning (first word) of the block transfer file.

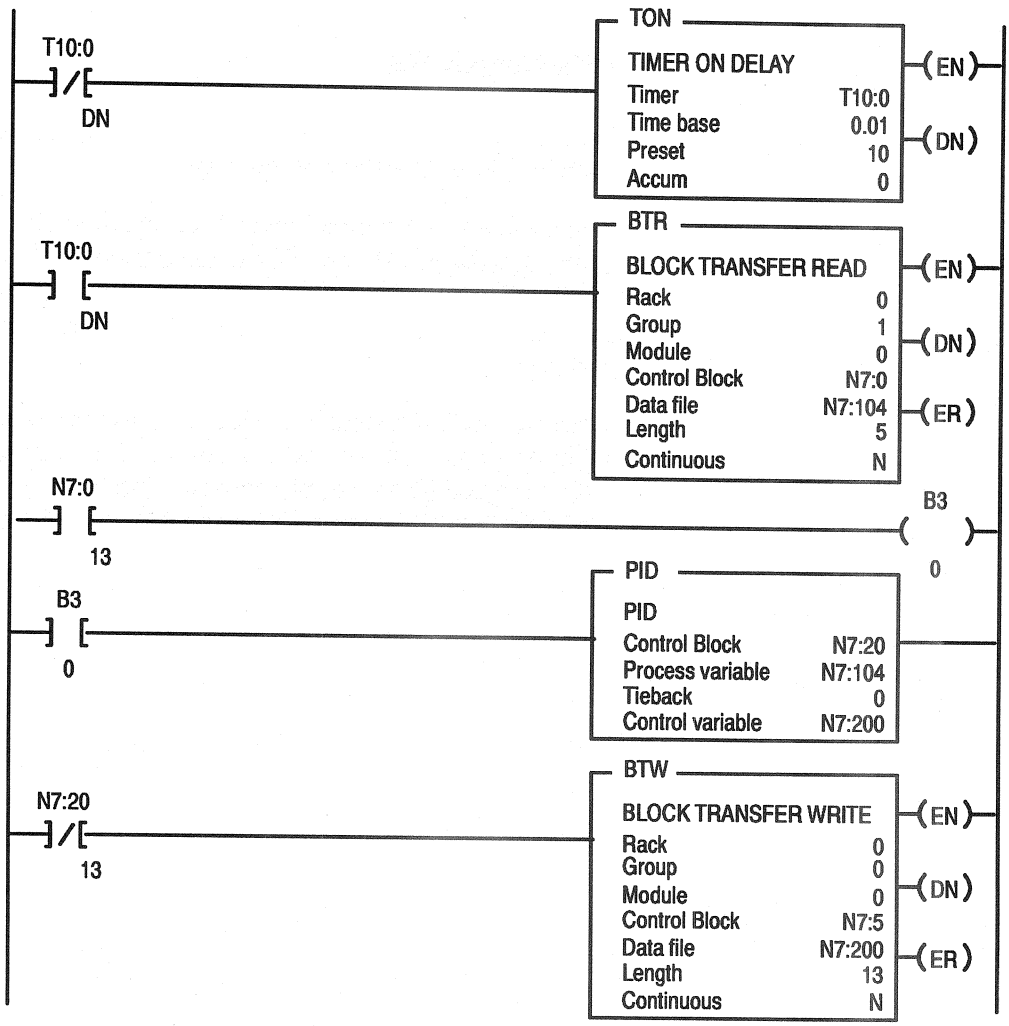
Main Program File

When you place the PID instruction in the main program file, control the sample time with a timer (Figure 14.7).

The accuracy of the timer depends on the time base and the total scan time of the processor. Always choose the 0.01 second time base for this PID application. Duplicate the timer instruction elsewhere in the program if the processor scan time (local I/O scan plus program scan) is greater than 2.5 seconds.

Because block transfers in the local chassis occur asynchronously during main program scan, you need a storage bit to ensure that the state of the PID input condition remains constant during the entire program scan. Condition all PID instructions using this storage bit.

Figure 14.7
Example PID Programming Conditioned by a Timer in the Main Program



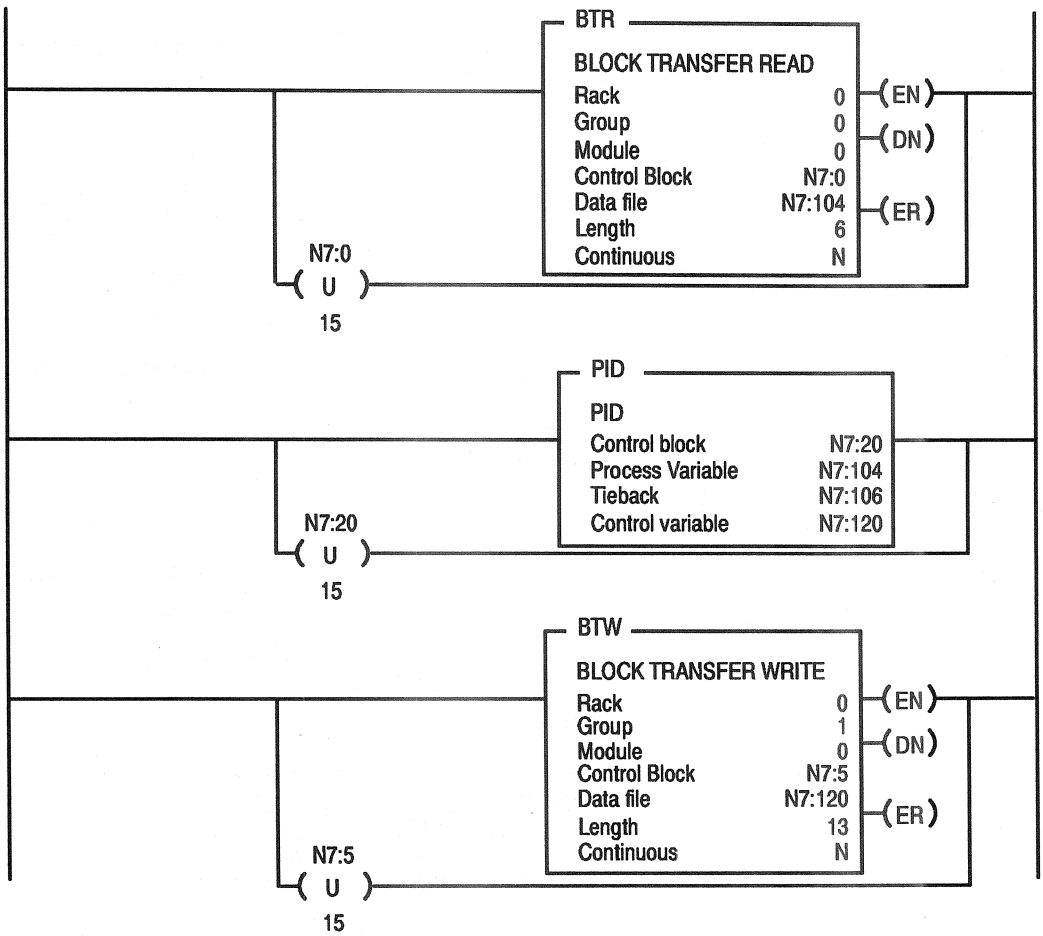
STI Program File

When you place the PID instruction in a selectable timed interrupt file, the STI controls the loop update (sampling) time (Figure 14.8).

The PID instruction operates on the most recent data when block transfer instructions are included in the STI file. You must place block transfer modules in the local chassis for this PID application. Unlatching the PID and BT enable bits forces the processor to run the PID and block transfer instructions every time the STI is enabled.

Important: The program scan waits for block transfer instructions in the STI file to complete their transfers.

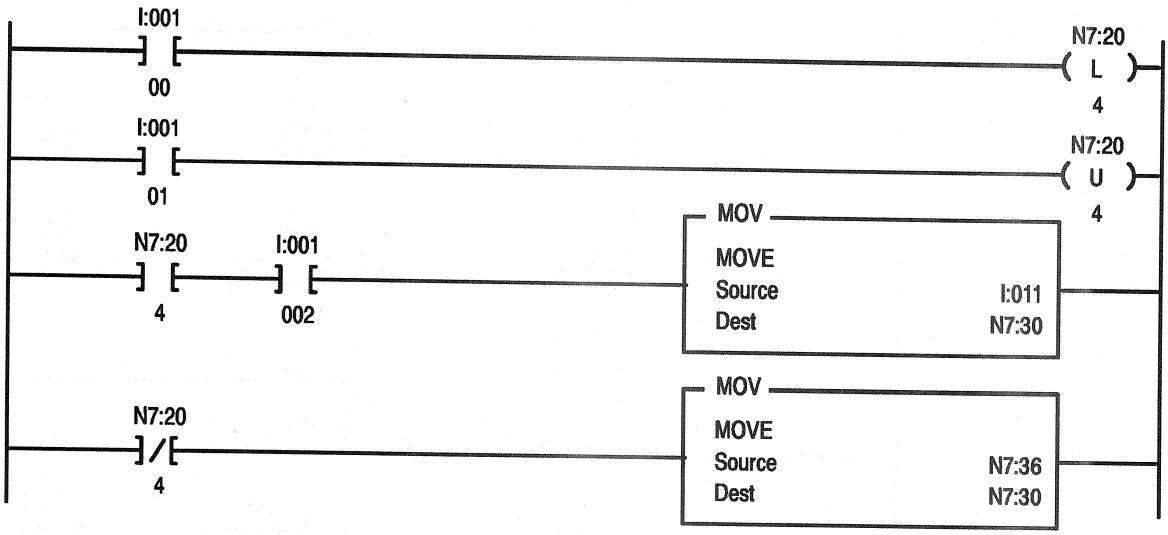
Figure 14.8
Example PID and Block Transfer Programming in an STI File



Ladder Logic Simulation of a Manual Control Station

When you program the simulation of a manual control station, make sure that a hardware manual control station is not connected when the program is enabled. Add the rungs in Figure 14.9 to the PID program in Figure 14.7 or Figure 14.8.

Figure 14.9
Example Program for Simulating a Manual Control Station



The last rung in the above example is for output tracking for bumpless transfer from automatic to manual mode.

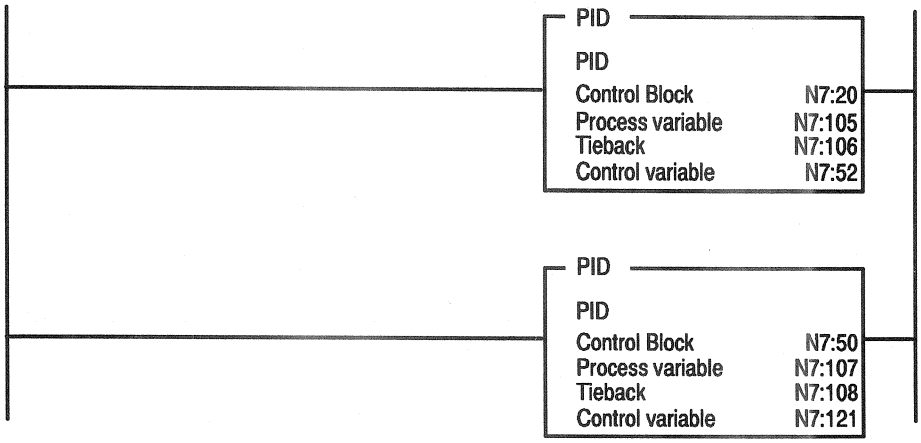
Address	Description
I:001/00	Manual pushutton switch
I:001/01	Automatic pushbutton switch
I:001/02	Enter pushbutton switch
I:011	Manual output value
N7:20/4	PID set output bit
N7:30	PID set output value
N7:36	Current control output

Cascading Loops

You can cascade two loops by assigning the control output of the outer loop to the setpoint of the inner loop. The setpoint of the inner loop is the third word (word 2) of the integer control block. If the control block of the inner loop is N7:50, address the outer loop control output at N7:52. Replace the PID rungs in Figure 14.7 or Figure 14.8 with those in Figure 14.10.

You must not scale the setpoint of the inner loop. Set the scaling bit (word 0, bit 5) to 1 to inhibit setpoint scaling.

Figure 14.10
Cascaded Loops



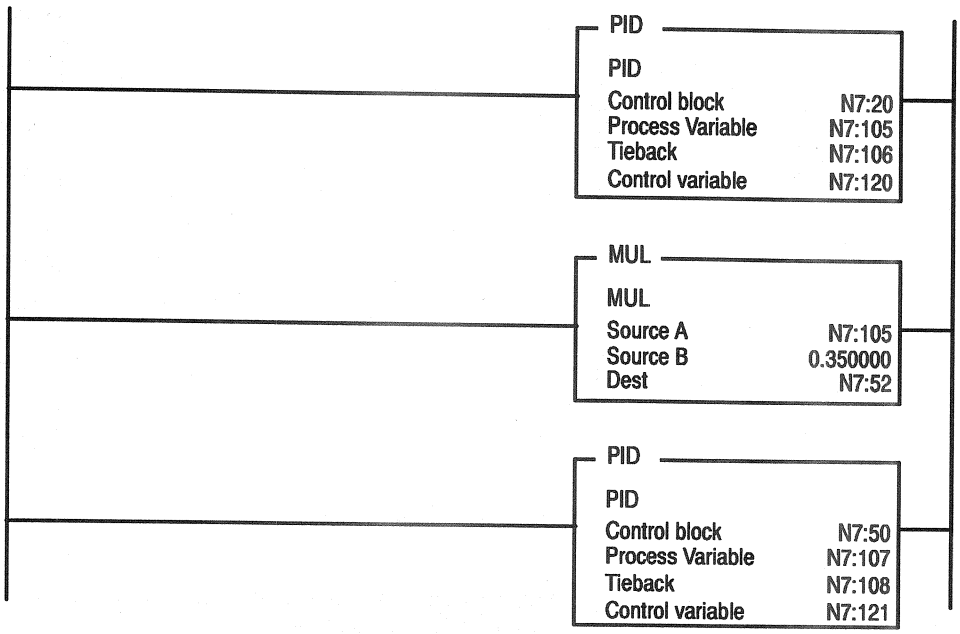
Ratio Control

You can maintain two values in a ratio by using a MUL instruction. Three parameters are involved:

- the wild or uncontrolled value
- the controlled value
- the ratio between these two values

Enter the address of the controlled value as the Destination. Enter the address of the wild or uncontrolled value as Source A. Enter either the address of the ratio value or a program constant for the ratio as Source B. For example, add the rungs in Figure 14.11 to the PID program in Figure 14.7 or Figure 14.8.

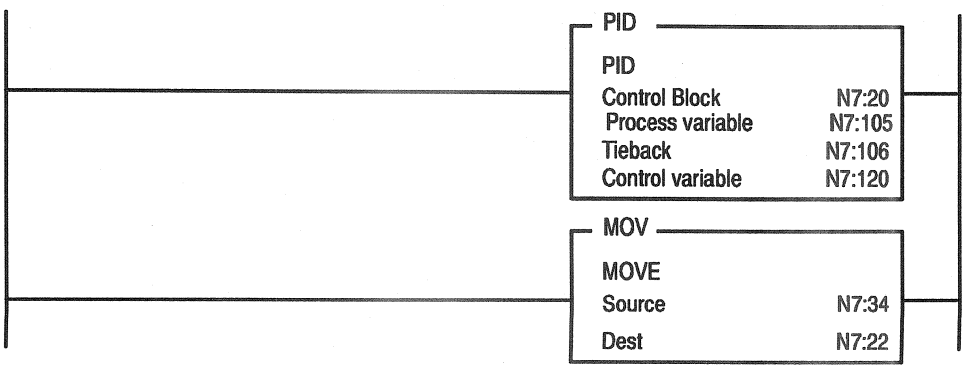
Figure 14.11
Ratio Control with a PID Instruction



Process Variable Tracking

When in manual control, your program can force the setpoint to be equal to the process variable (PV) by moving the process variable into the setpoint word (word 2 of the integer control block) to achieve a smooth manual-to-automatic transfer. If the setpoint is scaled, move the scaled PV from the PID control block directly into the setpoint word. If the setpoint is not scaled, move the unscaled value from the PV address in the PID instruction to the setpoint. For an example, add the rungs in Figure 14.12 to the PID program in Figure 14.7 or Figure 14.8.

Figure 14.12
Process Variable Tracking



Chapter 15

Block-Transfer Instructions BTR and BTW

Using Block Transfer Instructions

With block-transfer instructions, you can transfer up to 64 words at a time to or from a block transfer module in a local or remote I/O chassis. You can also transfer up to 64 words at a time to or from a supervisory processor when the processor is configured for adapter mode.

The PLC-5/30, -5/40 and -5/60 processors have configurable communication channels; choose between remote I/O scanner, remote I/O adapter, or DH+. Ladder block transfer instructions are not necessary when using adapter mode.

Table 15.A describes how to block transfer data to a local or remote rack when the processor is configured for scanner mode. Figure 15.1 illustrates how the transfer occurs.

Table 15.A
Block Transfer Instructions for Local or Remote Racks in Scanner Mode

If You Want to Transfer Data:	Use:
To the BT I/O module	BTW (block-transfer write)
From the BT I/O module	BTR (block-transfer read)

Figure 15.1
Block Transfer Operation in Scanner Mode

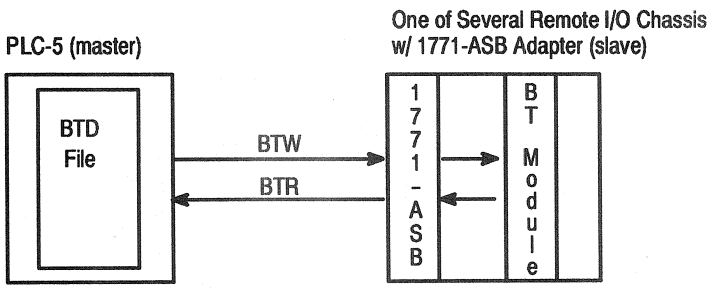
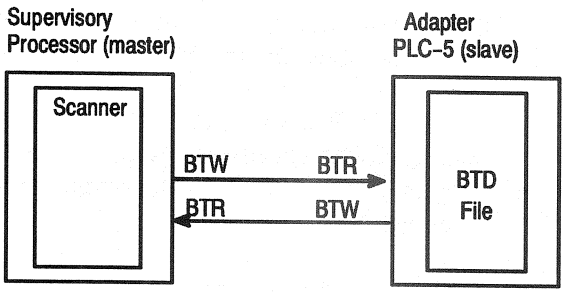


Table 15.B describes how to block transfer data when the processor is configured for adapter mode. Figure 15.2 illustrates how the transfer occurs.

Table 15.B
Block Transfer Instructions for Adapter Mode

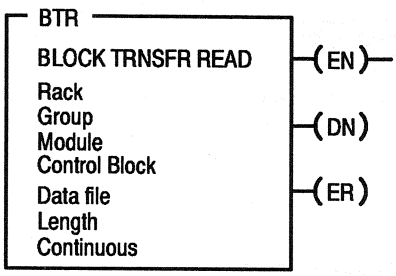
If You Want to Transfer Data:	Use:
From the supervisory processor	BTR (block-transfer read)
To the supervisory processor	BTW (block-transfer write)

Figure 15.2
Block Transfer Operation in Adapter Mode



NOTE: Both processors simultaneously execute the opposite block transfer instruction.

Block-Transfer Read (BTR) and Block-Transfer Write (BTW)



Description:
When the rung goes true, the BTW instruction tell the processor to write data stored in the data file to the specified rack/group/module address; the BTR instruction tells the processor to read data from the rack/group/module address and store it in the data file.

Block-Transfer Request Queue

When a false-to-true rung transition enables a BTW or BTR instruction, the transfer request is queued:

- PLC-5/10, -5/12, -5/15, -5/25 processors: the queue can hold up to 17 block transfer requests per logical rack
- PLC-5/30 processors: the queue can hold up to 128 block transfer requests to remote racks (maximum 64 per channel pair – 1A/1B); there is no limit for requests to local racks
- PLC-5/40, -5/60 processors: the queue can hold up to 128 block transfer requests to remote racks (maximum 64 per channel pair – 1A/1B, 2A/2B); there is no limit for requests to local racks

The processor then runs each block transfer request in the order they are scanned. However, when the processor is changed to Program mode, the block transfers still in the buffer are cancelled.

For PLC-5/10, -5/12, -5/15, -5/25 processors, each rack number has a block transfer queue with a corresponding queue-full bit. Table 15.C lists the queue-full bits. Once these bits are set, your ladder program must clear them. Your program should continually monitor these queue-full bits, found in the status file, word 7, bits 08–15. (Note: Since the PLC-5/30, -5/40 and -5/60 processors can have unlimited block transfers in local racks, there are no queue-full bits.)

Table 15.C
Queue-Full Bits for Block Transfer Requests (Word 7)

Bit	Description
08	Block-transfer queue for rack 0 is full
09	Block-transfer queue for rack 1 is full
10	Block-transfer queue for rack 2 is full
11	Block-transfer queue for rack 3 is full
12	Block-transfer queue for rack 4 is full
13	Block-transfer queue for rack 5 is full
14	Block-transfer queue for rack 6 is full
15	Block-transfer queue for rack 7 is full

The number of racks in your system depends on the processor you use.

A BTR or BTW instruction writes values into its control block address (a five-word integer file) when the instruction is entered. The processor uses these values to execute the transfer.

The PLC-5/30, -5/40 and -5/60 processors also have a block transfer file type (BT). You can still use existing programs with integer file types, but the new BT file type makes addressing easier. For example, if you need two control files, you can use BT10:0 and BT10:1; using integer files, you would have to use, for example, N7:0 and N7:5.

Entering Parameters

To program a BTW or BTR instruction, you must provide the processor with the following information that it stores in its control block:

- **Rack** is the I/O rack number (00-27 octal) of the I/O chassis in which you placed the target I/O module. Table 15.D lists the valid ranges for rack numbers.

Table 15.D
Valid Ranges for Rack Number in Block Transfer Instructions

Processor	Maximum Racks	Valid Range for Rack Numbers (octal)
PLC-5/10, -5/12, -5/15	4	00-03
PLC-5/25, -5/30	8	00-07
PLC-5/40, -5/40L	16	00-17
PLC-5/60, -5/60L	24	00-27

- **Group** is the I/O group number (0-7) which specifies the position of the target I/O module in the I/O chassis.
- **Module** is the slot number (0-1) within the group. The 0 slot is the low slot; the 1 slot is the high slot. You should use 0 for the module when using 1- or 1/2-slot addressing.
- **Control Block** is a block transfer control file (BT) or a five-word integer file (N) that controls the instruction's operation. Enter this file address without the # symbol. This is not a control file (type R).

  Important: In a PLC-5/40 or -5/60 processor, the block transfer data type (BT) must be used for rack addresses greater than 7.
- **Data File** is the address of the integer data file (N) from which (write) or into which (read) the processor transfers data. Enter this file address without the # symbol.
- **Length** is the number of words to transfer.

If You Set the Length to:	The Processor:
0	Reserves 64 words for block transfer data. The block transfer module transfers the maximum words it can handle.
1 to 64	Transfers the number of words specified.

- **Continuous** determines the mode of operation.

If You Specify:	The Instruction Uses This Mode:
Yes	Continuous - once the rung goes true, the instruction continues to transfer data until the continuous (.CO) bit is reset and the rung is false or you edit the instruction and specify NO for continuous mode.
No	Non-continuous - the instruction is enabled each time the rung goes true and performs only one data transfer per rung transition.

Using Status Bits

To use the BTR and BTW instructions correctly, the ladder program must examine the instruction's status bits stored in the control block. These bits are in word 0 of the control block.



WARNING: Except for the continuous bit .CO (bit 11) and the timeout bit .TO (bit 08), do not modify any status bit while the block transfer instruction is enabled. Unpredictable operation could occur with possible damage to equipment and/or injury to personnel.

Important: The bit labels (.EN, .ST, .CO, etc.) can only be used with the block transfer file type (BT).

This Bit:	Is Set:
Enable .EN (bit 15)	when the rung goes true. This bit shows that the instruction is enabled (that the block transfer is in progress). In non-continuous mode, .EN bit remains set until the block transfer finishes or fails and the rung goes false. In continuous mode, once .EN bit is set, it remains set regardless of the rung condition.
Start .ST (bit 14)	when the processor begins transferring data. The .ST bit is reset at the false-to-true transition after the .DN bit or .ER bit is set.
Done .DN (bit 13)	at completion of the block transfer, if the data is valid. The .DN bit is reset the next time the associated rung goes from false to true.
Error .ER (bit 12)	when the processor detects that the block transfer failed. The .ER bit is reset the next time the associated rung goes from false to true.
Continue .CO (bit 11)	when you edit the instruction for repeated operation of the block transfer after the first scan, independent of whether the processor continues to scan the rung Reset the .CO bit if you want the rung condition to initiate block transfers (return to non-continuous mode). If you are using continuous block transfers in a sequential function chart, see Appendix B, "SFC Instruction Timing and Memory Requirements" in this module.
Enable-waiting .EW (bit 10)	when the block transfer request enters the queue. If the queue is full, this bit remains reset until there is room in the queue. The .EW bit is reset when the associated rung goes from false to true. In continuous mode, once the .EW bit it set, it remains set. Use the .EW bit to verify that a BTW or BTR instruction is queued before leaving an SFC step.
No Response .NR (bit 09)	if the block transfer module does not respond to the first block transfer request. The .NR bit is reset when the associated rung goes from false to true.

This Bit:	Is Set:
Time Out .TO (bit 08)	if you reset the time out bit through ladder logic, the processor repeatedly tries to send a block transfer request to an unresponsive module for four seconds before setting the .ER bit. If you set the .TO bit through ladder logic, the processor disables the four-second timer and requests a block transfer one more time before setting the .ER bit. Note that this bit applies only to local and extended local I/O block transfers.
Read-Write .RW (bit 07)	controlled by the instruction. A 0 indicates a write operation; a 1 indicates a read operation.



WARNING: The processor runs block transfer instructions asynchronously to program scan. The status of these bits could change at any point in the program scan. If you examine these bits in ladder logic, copy the status once to a storage bit whose status is synchronized with the program scan. Otherwise, timing problems may invalidate your program with possible damage to equipment or injury to personnel.

Your ladder program should condition the use of block transfer data on the state of the .ER bit.

Using the Control Block

In addition to the status bits, the control block contains other parameters the processor uses to control block transfer instructions. Table 15.E lists these values.

Table 15.E
Values in the Control Word

Word – Integer Control Bock	BT Control Block	Description
0	.EN thru .RW	status bits
1	.RLEN	Requested word count
2	.DLEN	Transmitted word count / Error Code (PLC-5/30, -5/40, -5/60)
3	.FILE	File type number
4	.ELEM	Element Number

Requested Word Count

This is the number of words to be transferred between the processor and the module (0–64 words); the processor creates a file of the length you specified that starts at the data address you enter. The length depends on the target module or your application. For example, if you specify 30 in this field, you are specifying a block length of 30 and the processor creates a 30-word file; if you specify 64, you are specifying a block length of 64 and the processor creates a 64-word file. If you specify a 0 when you enter the block transfer instruction, the processor lets the block transfer module determine the number of words that need to be transferred and creates a default 64-word file.

Transmitted Word Count

This is the number of words the module actually transferred after the instruction completes exectution. The processor uses this number to verify the transfer. This number should match the requested word count (unless the transmitted word count is zero). If these numbers do not match, the processor sets the .ER bit (bit 12).

Error Code (PLC-5/30, -5/40, -5/60 Processors)

The PLC-5/30, -5/40 and -5/60 processors also have error codes (word 2 of the integer file control block or stored in the .DLEN word of the BT control block) that the processor can set during the transfer. If a block transfer error occurs in a PLC-5/30, -5/40 or -5/60, the error code is stored in the transmitted word count. This error can be identified by its negative number. Only one error code is stored at a time (a new error code overwrites any previous error code). Table 15.F lists these error codes.

Table 15.F
PLC-5/30, -5/40, -5/60 Block Transfer Error Codes

Error Number:	Description:
-1	not used
-2	not used
-3	The size of the block transfer plus the size of the index in the block transfer data table was greater than the size of the block transfer data table file.
-4	There was an invalid transfer of block transfer write data between the adapter and the block transfer module.
-5	The checksum of the block transfer read data was wrong.
-6	The block transfer module requested a different length than the associated block transfer instruction. This could happen if a 64-word block transfer instruction was executed and the block transfer module's default length was not 64 words.
-7	Block transfer data was lost due to a bad communication channel. Possible reasons are noise, bad connections, and loose wires.
-8	Error in block transfer protocol - unsolicited block transfer
-9	The block transfer timeout, set in the instruction, timed out before completion.
-10	No communication channels are configured for remote I/O.
-11	No communication channels are configured for the requested rack or slot.
-12	not used
-13	Queues for remote block transfers are full.

File Number

This number identifies the file number of the integer file from which the data is written or to which the data is read. For example, the file number of N7:20 is 7.

Element Number

This number identifies the starting word in the data file address. For example, in N7:20, the word number is 20.

Selecting Continuous Operation

Continuous block transfer is similar to discrete I/O transfer in that the I/O is updated continuously, but continuous block transfer updates block transfer I/O, such as analog input and analog output data. A continuous block transfer maintains block integrity (i.e., only full words are updated, not partial words).

Continuous mode lets you perform multiple block transfers by programming only one block transfer instruction (with no input conditions on the rung). Once the continuous block transfer starts, the transfer is continuously executed once per scan, independent of whether the processor continues to scan the associated rung and independent of the rung condition. To enable continuous operation, select Continuous when you enter the block transfer instruction.

Continuous mode works as follows (Figure 15.3):

1. When the rung that contains the block transfer instruction goes true, the processor sets the .EN bit. The processor also resets the .DN, .ER, .ST, .EW and .NR bits.

2. The processor then queues the block transfer request. When the block transfer request enters the queue, the processor sets the .EW bit.

3. When the processor starts to process the block transfer request, the processor sets the .ST bit.

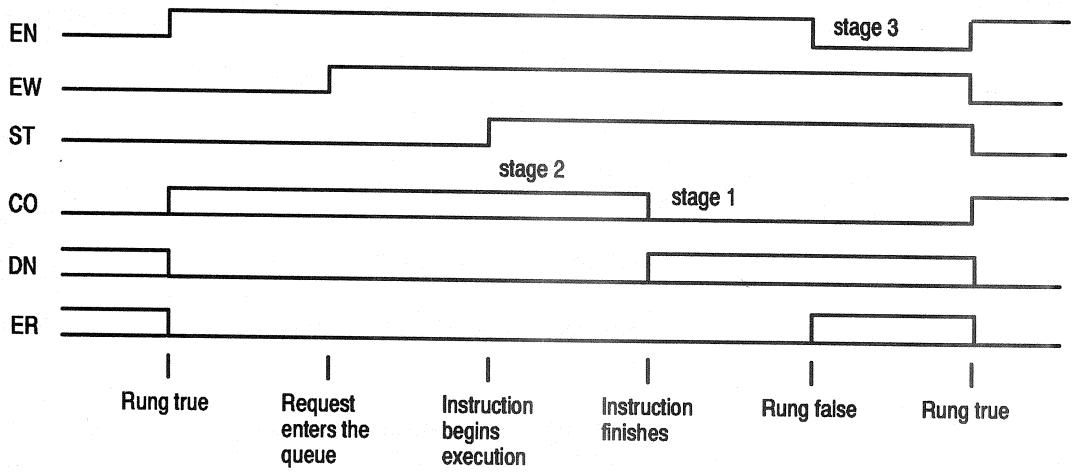
4. If no error occurs during transmission, the processor sets the .DN bit. The processor copies the actual number of elements sent or received by the block transfer instruction into the transmitted word count (word 2 of the control block).

   If an error occurs in a PLC-5/10, -5/12, -5/15, or -5/25, the processor sets the .ER bit; if an error occurs in a PLC-5/30, -5/40 or -5/60, the processor also puts the error code in the transmitted word count location as a negative number.

5. If there is no response (and after the processor sets the .NR bit), the processor tries to send the block transfer again. If the .TO bit is reset, the processor repeatedly sends the request for four seconds. If the .TO bit is set, the processor only retries the request one time.

6. If a continuous block transfer has an error, you must restart it to continue. (See Figure 15.6 on page 15-16 for an example program.)

Figure 15.3
Timing Diagram for Status Bits in Continuous BTR and BTW Instructions



Stage 1 - If .CO set, return to stage 2; if .CO reset, go to stage 3
Stage 2 - Return here for continuous operation
Stage 3 - Go here if .CO is reset

A continuous block transfer continues as long as the processor stays in Run or Test mode and the transfer does not error. If you switch to Program mode or the processor faults, the block transfer stops and will not start again until the processor scans the rung that contains the block transfer instruction. If running continuous block transfers from within sequential function charts, see Appendix B, "SFC Instruction Timing and Memory Requirements" in this module.

To stop continuous operation either: modify the block transfer instruction and select non-continuous, or reset the .CO bit.

Selecting Non-Continuous Operation

Non-continuous block transfer updates block transfer I/O one time when the rung goes true. A non-continuous block transfer maintains block integrity. The entire block of data is updated each time the processor runs the block transfer instruction. Use non-continuous mode when you want to control when the block transfer occurs or the number of times the block transfer occurs.

Non-continuous mode works as follows (Figure 15.4):

1. When the rung that contains the block transfer instruction goes true, the processor sets the .EN bit. The processor also resets the .DN, .ER, .ST, .EW and .NR bits.

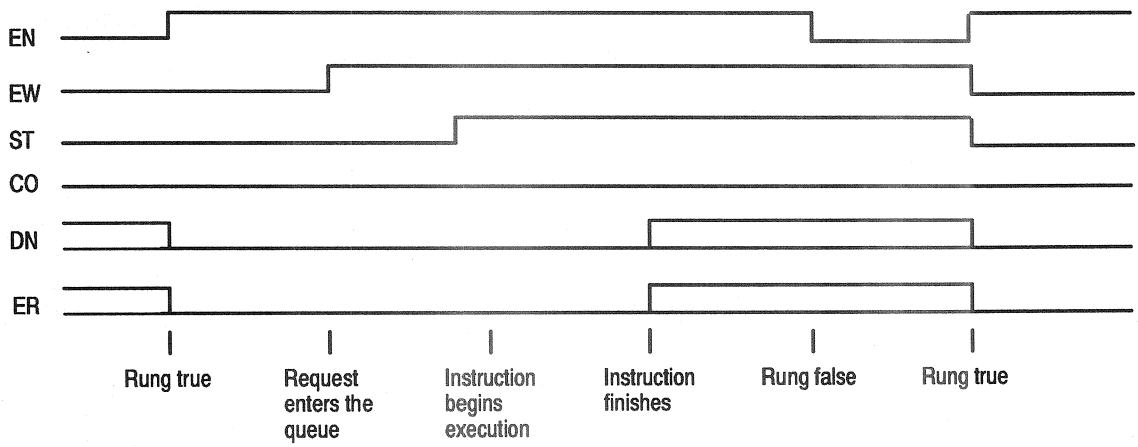
2. The processor then queues the block transfer request. When the block transfer request enters the queue, the processor sets the .EW bit.

3. When the processor starts to process the block transfer request, the processor sets the .ST bit.

4. If no error occurs during transmission, the processor sets the .DN bit after the block transfer instruction finishes. If an error occurs, the processor sets the .ER bit.

5. This signifies that one block transfer finished. The next time the rung goes false, the processor resets the .EN bit.

Figure 15.4
Timing Diagram for Status Bits in Non-Continuous BTR and BTW Instructions



Block Transfer Timing PLC-5/10, -5/12, -5/15, -5/25

The time to complete a block transfer in a PLC-5/10, -5/12, -5/15, -5/25 processor depends on:

- instruction run time
- waiting time in the queue
- transfer time

Instruction Run Time

The time in microseconds it takes the processor to execute a block transfer instruction is defined by these formulas:

Write:

```
310 + 11.2Q + 5.4W
```

Read:

```
250 + 11.2Q
```

Where:	Represents:
Q	number of queued block transfer requests to the same I/O chassis with the continuous bit set
W	number of words to transfer

Waiting Time in the Queue

The waiting time in the queue is the sum of the transfer times yet to occur before the block transfer request (for which you are calculating time) to the same I/O chassis.

Transfer Time

The transfer time in milliseconds between the active buffer and the module starts when the processor sets the start bit and ends when the processor sets the done bit. The transfer time is defined by these formulas:

Write:

local `0.9 + 0.1W`

remote (57.6K baud) `13 + 30C + 0.3W`

Read:

local `0.9 + 0.1W`

remote (57.6K baud) `9 + 21.3C + 0.3W`

Where:	Represents:
C	number of remote I/O chassis
W	number of words to transfer

Block Transfer Timing PLC-5/30, -5/40, -5/60

The time to complete a block transfer in a PLC-5/30, -5/40 or -5/60 processor depends on:

- instruction run time
- waiting time in the holding area (queue)
- transfer time

Instruction Run Time

The time it takes the processor to execute a block transfer instruction is the same for a read or a write: 450 microseconds.

Waiting Time in the Holding Area

The waiting time in the holding area is the sum of the transfer times yet to occur before the block transfer request (for which you are calculating time) to the same I/O chassis.

Transfer Time

The transfer time in milliseconds between the active buffer and the module starts when the processor sets the start bit and ends when the processor sets the done bit. The transfer time is defined by this formula (same for a read or write):

local 600 µsec + *x*(w)

remote (57.6K baud) `4 + 8C + 0.3W`

remote (115K baud) `4 + 4.6C + 0.15W`

Where this:	Represents:
x	• 8 or less block transfers queued in local rack = 86 µsec • more than 8 block transfers queued in local rack = 300 µsec NOTE: This timing assumes that no other block transfers are queued to the same slot and that successive block transfers to the same slot are executed every 1000 µsec.
C	number of remote I/O chassis
W	number of words to transfer

Programming Examples

Program your processor for block transfer using one of the following methods based on your application requirements (Table 15.G):

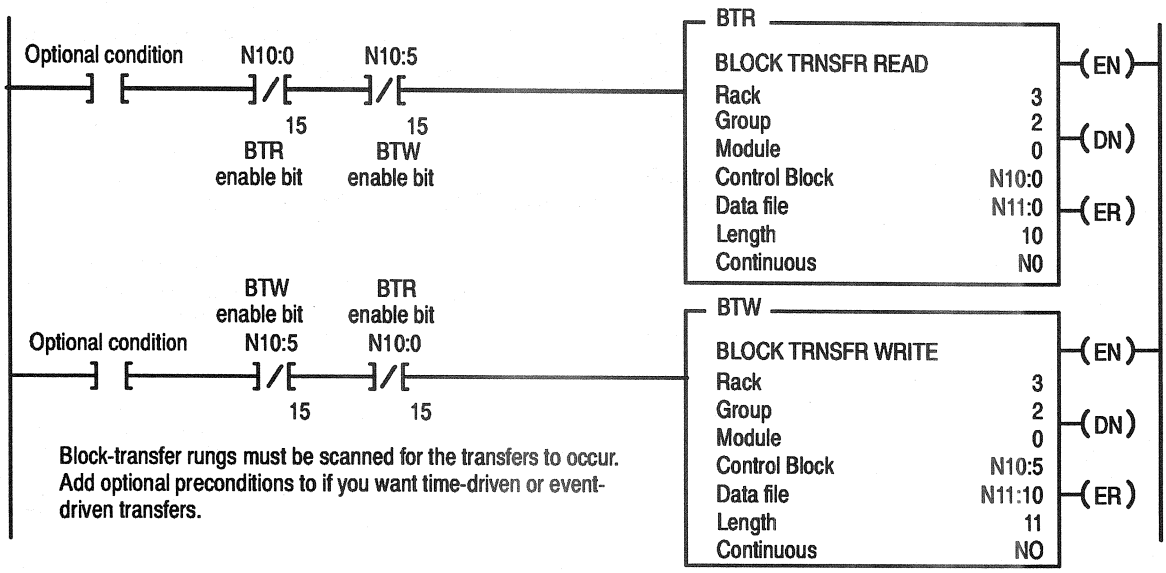
Table 15.G
Block Transfer Programming Methods

If You Want to:	Use This Method:
Program block transfers to and from the same module when you want the order of execution to follow the same order scanned in the program Ensure block integrity	Bidirectional alternating
Program block transfers to and from the same module when you want the transfers to continue regardless of which SFC steps are active	Bidirectional continuous
Program a BTR from or a BTW to a module when you want the order of execution to follow the order scanned in the program Ensure block integrity	Directional non-continuous
Program a BTR from or BTW to a module when you want the transfer to continue regardless of which SFC steps are active	Directional continuous

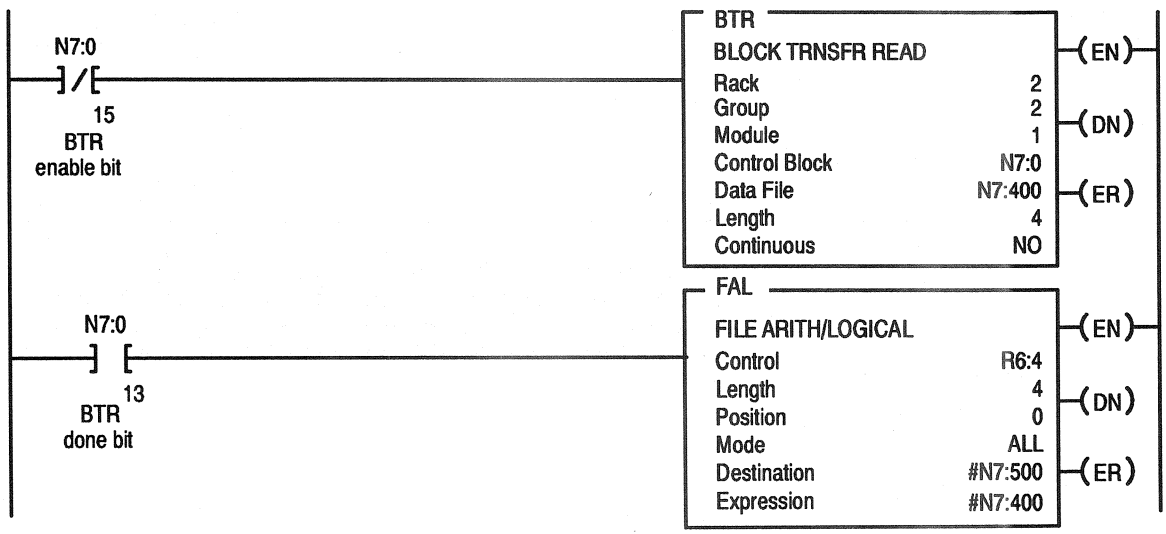
Example Bidirectional Alternating Block Transfer

Figure 15.5 shows a bidirectional alternating block transfer example. Using rungs like this example makes sure the block transfer requests are executed in the order in which they were sent to the queue. The processor alternates between the BTRs and BTWs in the order in which they are scanned. The block transfer continues as long as the optional condition is true.

Figure 15.5
Example Bidirectional Alternating Block Transfer



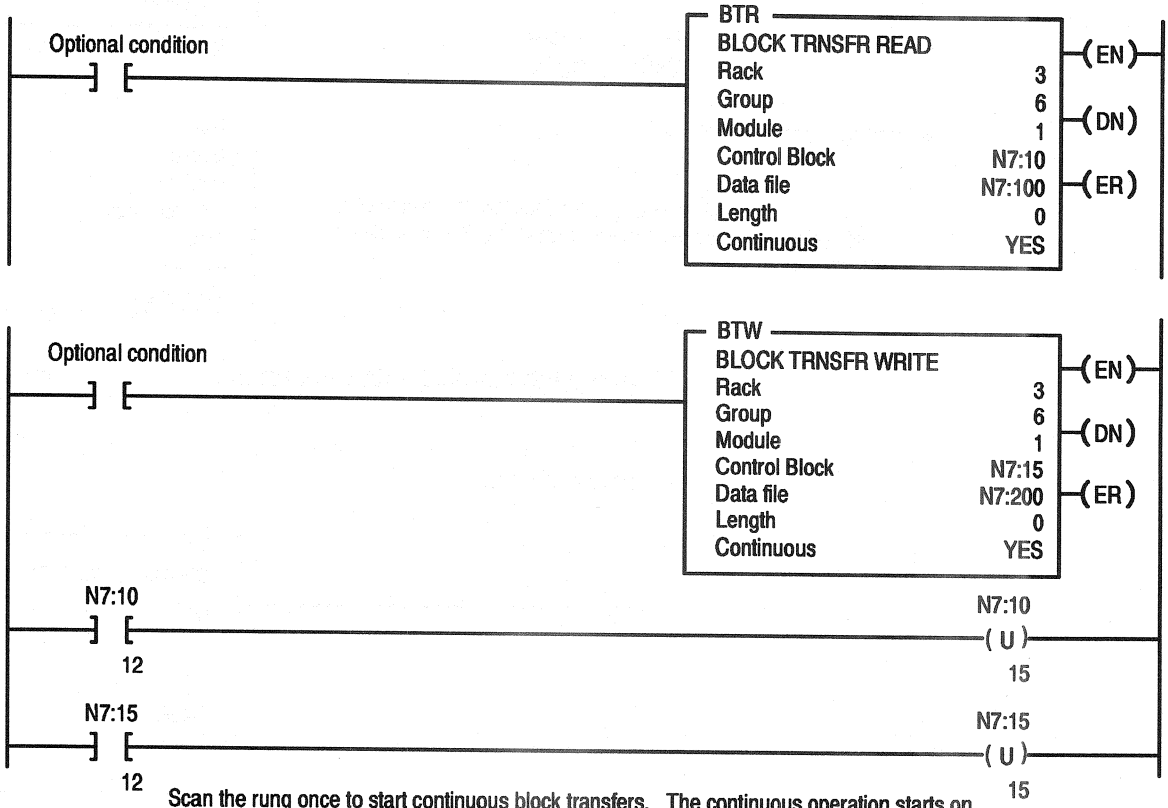
If you block transfer data continuously, we recommend that you buffer the file by examining the BTR done bit and executing a file-to-file move (or copy) when the done bit is true (to ensure file integrity of the block transfer read data file):



Example Bidirectional Continuous Block Transfer

Data storage is asynchronous. The transfer can contain data that was stored in two different I/O updates. Figure 15.6 shows a bidirectional continuous block transfer example.

Figure 15.6
Example Bidirectional Continuous Block Transfer



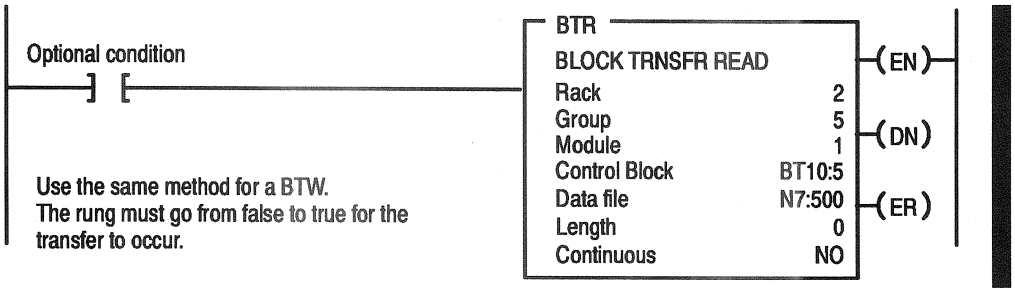
Scan the rung once to start continuous block transfers. The continuous operation starts on a false-to-true rung transition and continues, whether or not the rungs are scanned. If an error occurs, the last two rungs re-start continuous operation by unlatching the control block error bits. To stop continuous operation, use the Data Monitor to reset the continuous bit (.CO or bit 11), or change the Continuous field in the instruction to NO.

Example Directional Non-Continuous Block Transfer

Figure 15.7 shows a directional non-continuous block transfer example.

Figure 15.7
Example Directional Non-Continuous Block Transfer

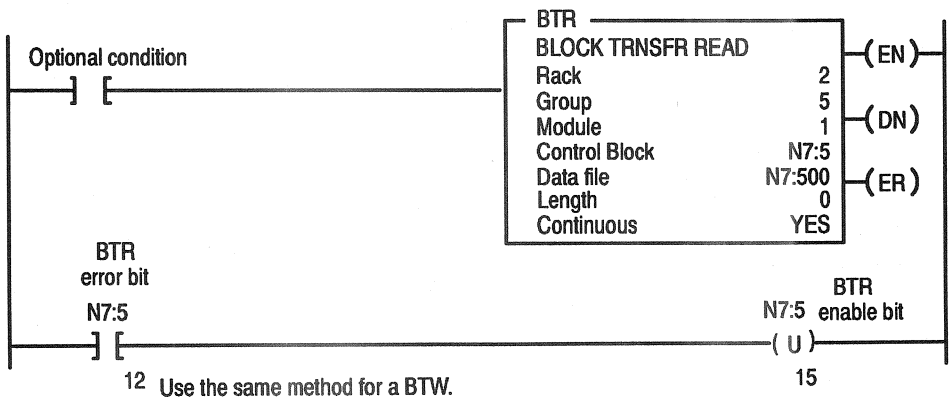
Important: This example shows a PLC-5/30, -5/40 or -5/60 using the BT file type. If you are using a PLC-5/10, -5/12, -5/15, or -5/25, substitute an appropriate integer file.



Example Directional Continuous Block Transfer

Data storage is asynchronous. The transfer can contain data that was stored in two different I/O updates. Figure 15.8 shows a directional continuous block transfer example.

Figure 15.8
Example Directional Continuous Block Transfer



Chapter 16

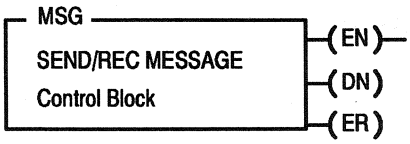
Message Instruction MSG

Using the Message Instruction

The message instruction (MSG) transfers data to and from processors on a Data Highway Plus link. You program the MSG instruction in the ladder logic of the processor that initiates the MSG instruction.

The MSG instruction can communicate with PLC-2, PLC-3, PLC-5, and PLC-5/250 processors on local or remote links.

Message (MSG)



Description:
The MSG instruction transfers up to 1000 elements of data; the size of each element depends on the data table section you specify and the type of message command you use. For example, one binary element contains one 16-bit word and one floating point element contains 2 words.

The MSG instruction transfers data in packets. Each packet can contain up to 120 words. If your message transfer contains more words than fit in one packet, the transfer requires more than one packet of transfer data. The more packets of data to transfer, the longer the transfer takes.

Entering Parameters

You specify a control block address when you first enter the MSG instruction. Then the programming terminal automatically displays a data entry screen, from which you enter instruction parameters that are stored at the control block address. You can also use the data monitor screen for the MSG instruction to edit selected parameters.

Control Block Address
With PLC-5/10, -5/12, -5/15 or -5/25 processors, use an integer file (N) without the # symbol for the message control block. For example N7:0 is a valid MSG control block address.

Note that the control block size varies according to the length of your message; if you are communicating with a PLC-2 processor, the control file will be approximately 11 or 12 words. If you are communicating with a PLC-3, PLC-5 or PLC-5/250 processor, the control file will be approximately 11 to 15 words. This size is displayed on the screen in the `BLOCK SIZE` field.

For PLC-5/30, -5/40 or -5/60 processors, you can use an integer file or the message (MG) file type to access the message control block. For example, MG10:0 is a valid MSG control block address for PLC-5/30, -5/40 or -5/60 processors. Using the MG file type, note that the control block size is fixed at 56 words; this size is displayed on the screen in the BLOCK SIZE field.

MSG Data Entry Screen

After you enter the control block address for a MSG instruction, the programming software automatically displays the MSG Data Entry screen for the appropriate data type.

If You Are Using This Data File Type:	See:
integer (N)	Figure 16.1
message (MG)	Figure 16.2

Figure 16.1
MSG Instruction Data Entry Screen (Integer Control Block)

```
MESSAGE INSTRUCTION DATA ENTRY FOR CONTROL BLOCK N7:10


            Communication Command              PLC-5 Typed Write
            PLC-5 Data Table Address:          undefined
            Size in Elements:                  1
            Local/Remote:                      LOCAL
               Remote Station:                 N/A
               Link ID:                        N/A
               Remote Link Type:               N/A
            Local Node Address:                00
            Destination Data Table Address:    undefined


 BLOCK SIZE IS undefined


Press a key to change a parameter or <ENTER> to accept parameters.

Program     Forces:None       Edits:None                          PLC-5/15 File DRILL1
 Read/    PLC-5  Size in Local/  Remote    Link     Remote    Local  Destin
 Write  Address Elemnts Remote  Station     ID      Link      Node  Address
  F1       F2      F3      F4      F5       F6       F7       F8     F9
```

Figure 16.2
MSG Instruction Data Entry Screen (MG Control Block)

```
MESSAGE INSTRUCTION DATA ENTRY FOR CONTROL BLOCK MG10:10

            Communication Command              PLC-5 Typed Write
            PLC-5 Data Table Address:          undefined
            Size in Elements:                  1
            Local/Remote:                      LOCAL
               Remote Station:                 N/A
               Link ID:                        N/A
               Remote Link Type:               N/A
            Local Node Address:                00
            Destination Data Table Address:    undefined
            Port Number                         1A

BLOCK SIZE IS 56


Press a key to change a parameter or <ENTER> to accept parameters.

Program    Forces:None      Edits:None                    PLC-5/40 File DRILL1
 Read/    PLC-5  Size in Local/  Remote    Link    Remote   Local  Destin   Port
 Write  Address Elemnts Remote  Station     ID      Link    Node  Address   Number
  F1      F2      F3      F4      F5       F6      F7      F8      F9      F10
```

Press the function key for the data you want to modify. You can specify the following MSG parameters from the Data Entry screen:

F1 – Read/Write specifies whether the MSG instruction performs a read or write operation. Pressing this key toggles through PLC-5, PLC-3, PLC-2, and ASCII options.

When you specify the ASCII option (for use with co-processors only), the software displays a new screen for you to enter the text for your ASCII communications:

```
MESSAGE INSTRUCTION DATA ENTRY FOR CONTROL BLOCK MG10:0

            Communication Command:             ASCII
            Port Number:                       1A

  +-------------  Command Text  --------------+
  |                                           |
  |                                           |
  |                                           |
  |                                           |
  +-------------------------------------------+

  BLOCK SIZE = 56 WORDS

Press a key to change a parameter or <ENTER> to accept parameters.
>
Rem Prog   Forces:None      Edits:None                    5/40 File BATCH40
Command  Port   Command
 Type   Number   Text
  F1      F2      F3
```

1. Press **[F2] – Port Number**; type the port number (channel). Valid options are: 0, 1A, 1B, 2A, 2B, or 3A (if you have a co-processor) and press **[Enter]**.

2. Press `[F3] - Command Text`; type the text on the input line and press `[Enter]`.

   The text appears in the text window.

F2 – PLC-5 Address specifies the data file address of the processor containing the message instruction. If the MSG operation is write, this address is the starting word of the source file. If the MSG operation is read, this address is the starting word of the destination file.

F3 – Size in Elements specifies the number of elements (1–1000) in the data file.

F4 – Local/Remote LOCAL specifies that the message is sent to a device on the local DH+ link; REMOTE specifies that the message is sent through a DH+ bridge (DH, DH II, etc.) to another DH +link.

If you select REMOTE, the function keys `[F5] - Remote Station`, `[F6] - Link ID`, and `[F7] - Remote Link` are active.

F5 – Remote Station specifies the DH or DH II address (1–376 octal) of the target station.

PLC-2 and PLC-3 processors require communication adapter modules (1771–KA2 and 1775–KA, respectively) when they operate as stations on data highway. In these configurations, the remote station address is the address of the communication adapter module.

F6 – Link ID specifies the remote link where the processor you want to communicate with resides. The default is 0.

F7 – Remote Link toggles through data highway, DHII, and other as choices for what connects the remote link to the local DH+.

F8 – Local Node specifies the local station address on the DH+ (0–77 octal).

If you communicate with another processor on the local link, this address is the address of the target station on the local link.

If you communicate with a target station on a remote link, this address is the station number of the communication adapter module that bridges the data highway.

F9 – Destination Address specifies the starting address of the source or destination file in the target processor.

F10 – Port Number specifies the channel for message communications. Valid options are: 0, 1A (default), 1B, 2A, 2B.

Using Status Bits

The MSG instruction uses the following status bits:



WARNING: Do not modify any status bit while the instruction is enabled. Unpredictable operation could occur with possible damage to equipment and/or injury to personnel.

This Bit:	Is Set:
Enable .EN (bit 15)	when the rung goes true This bit shows that the instruction is enabled (that the instruction is being executed). In non-continuous mode, .EN bit remains set until the message is completed and the rung goes false. In continuous mode, once .EN bit is set, it remains set regardless of the rung condition.
Start .ST (bit 14)	when the processor begins executing the MSG instruction. The .ST bit is reset when the .DN bit or .ER bit is set.
Done .DN (bit 13)	when the last packet of the MSG instruction transferred. The .DN bit is reset the next time the associated rung goes from false to true. The .DN bit is only active in non-continuous mode.
Error .ER (bit 12)	when the processor detects that the message transfer failed. The .ER bit is reset the next time the associated rung goes from false to true.
Continue .CO (bit 11)	manually for repeated operation of the MSG instruction after the first scan, independent of whether the processor continues to scan the rung. Reset the .CO bit if you want the rung condition to initiate messages (return to non-continuous mode).
Enable-Waiting .EW (bit 10)	when the processor detects that a message request entered the queue. The processor resets the .EW bit when the .ST bit is set.
No Response .NR (bit 09)	if the target processor does not respond to the first MSG request. The .NR bit is reset when the associated rung goes from false to true.
Time Out .TO (bit 08)	If you set the .TO bit through ladder logic, the processor stops processing the message and sets the .ER bit (timeout error 55). A message will automatically time out in 30-60 seconds.



WARNING: The processor controls status bits .ST and .EW asynchronously to the program scan. If you examine these bits in ladder logic, copy the status to a storage bit whose status is synchronized with the program scan. Otherwise, timing problems may invalidate your program with possible damage to equipment or injury to personnel.

Monitoring a Message Instruction

To monitor or edit MSG instruction parameters and status bits after you enter the MSG instruction, display the data monitor screen for the MSG instruction.

If You Are Using This File Type:	See:
integer (N)	Figure 16.3
message (MG)	Figure 16.4

Figure 16.3
Data Monitor Screen for the MSG Instruction – Integer File Type

```
    MESSAGE INSTRUCTION DATA MONITOR FOR CONTROL BLOCK N7:10

 Communication Command              PLC-5 Typed Write
 PLC-5 Data Table Address:          I:000          ignore if timed-out:  0 TO
 Size in Elements:                  1                    to be retried:  0 NR
 Local/Remote:                      LOCAL              awaiting execution:  0 EW
    Remote Station:                 N/A                      continuous:  0 CO
    Link ID:                        N/A                           error:  0 ER
    Remote Link Type:               N/A                   message done:  0 DN
 Local Node Address:                00                 message transmitting:  0 ST
 Destination Data Table Address:    O:000                   message enabled:  0 EN
 Port Number                        1A
                                                  control bit addr:  N7:10/8
 ERROR CODE: 0

 BLOCK SIZE = 9 WORDS


Press a function key to change a value.

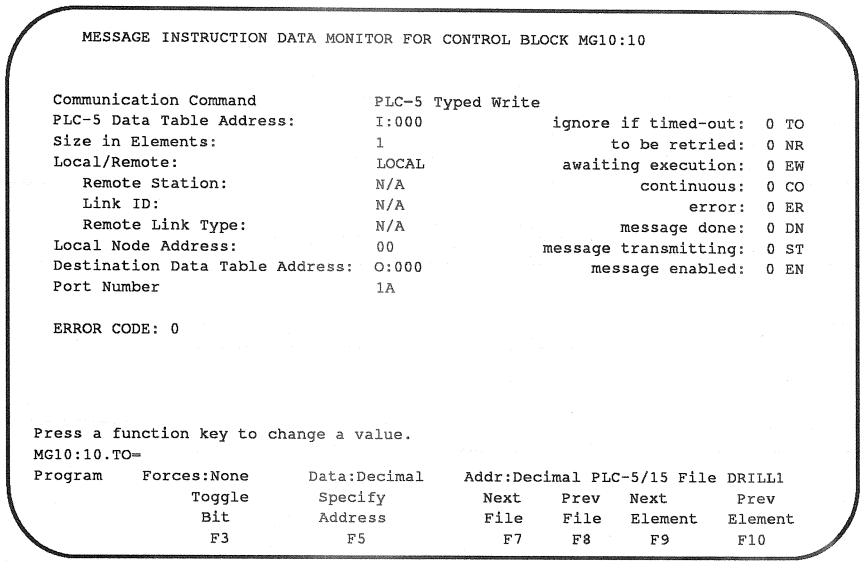
Program    Forces:None      Data:Decimal    Addr:Decimal PLC-5/15 File DRILL1
             Size in                                        Toggle
             Elemnts                                          Bit
               F3                                             F9
```

You can change the following information on this screen:

F3 – Size in Elements specifies the number of elements (1–1000) you want to read from or write to the network station.

F9 – Toggle Bit lets you set and reset status bits. Place the cursor on the status bit value and press `[F9]`.

Figure 16.4
Data Monitor Screen for the MSG Instruction - Message File Type



Selecting Continuous Operation

Continuous mode lets you use multiple message transfers by programming only one MSG instruction (with no input conditions on the rung). Once the continuous message transfer starts, the transfer is continuously executed, independent of whether the processor continues to scan the associated rung and independent of the rung condition. To enable continuous operation, set the .CO bit.

Continuous mode works as follows (Figure 16.5):

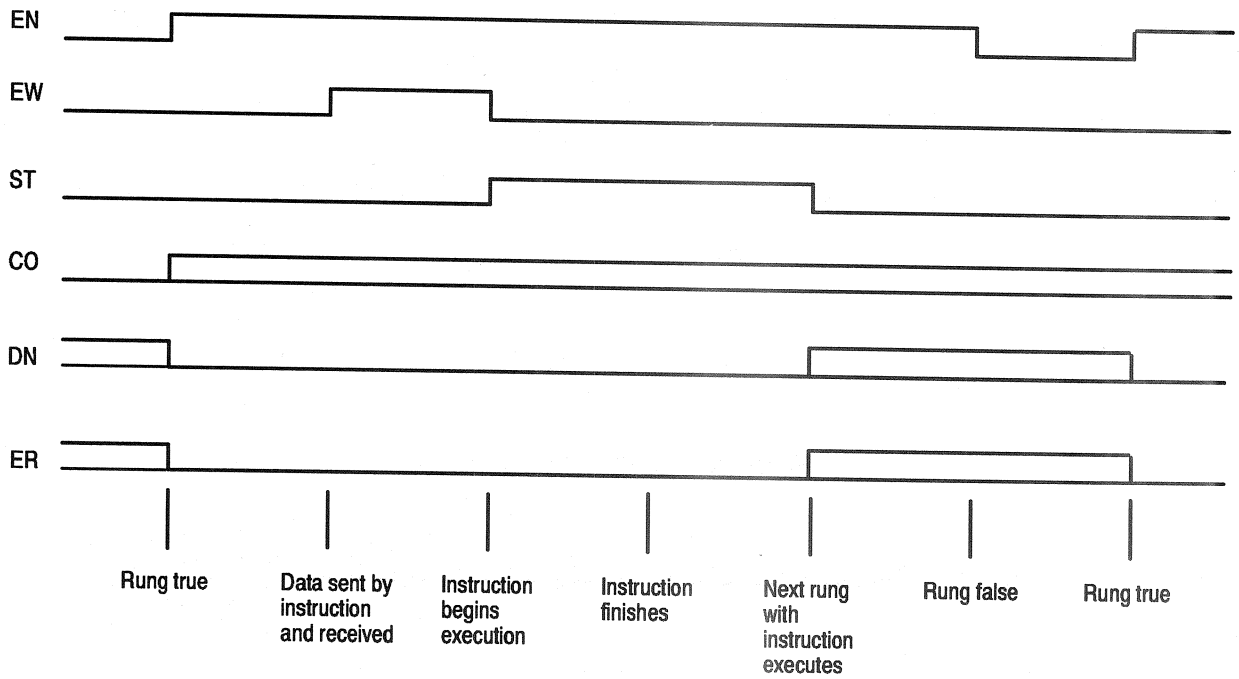
1. When the rung that contains the MSG instruction goes true, the processor initiating with the MSG instruction sets the .EN bit. The processor also resets the .DN and .ER bits.

2. The processor then queues the message request. When the message request enters the queue, the processor sets the .EW bit.

3. When the processor starts to process the message request, the processor sets the .ST bit. The next time the processor receives network control, the processor transmits the message.

4. If no errors occur during transmission, the processor sets the .DN bit after the last packet in the first execution of the MSG instruction transfers. If an error occurs, the processor sets the .ER bit and stores an error code in the lower byte of word 0 of the control block for PLC-5/10, -5/12, -5/15, -5/25 and word 1 of the control block for PLC-5/30, -5/40 and -5/60.

Figure 16.5
Timing Diagram for Status Bits in Continuous MSG Instructions



A continuous message transfer continues as long as the processor stays in Run or Test mode. If you switch to Program mode or the processor faults, the message transfer stops and will not start again until the processor scans the rung that contains the MSG instruction.

To stop continuous operation, reset the .CO bit.

Selecting Non-Continuous Operation

Non-continuous mode performs the message transfer one time for each false-to-true transition of the rung that contains the MSG instruction. Non-continuous operation occurs as long as the .CO bit remains reset. Use non-continuous mode when you want to control when the message transfer occurs or the number of times the message transfer occurs.

Non-continuous mode works as follows (Figure 16.6):

1. When the rung that contains the MSG instruction goes true, the processor initiating the MSG instruction sets the .EN bit. The processor also resets the .DN and .ER bits.

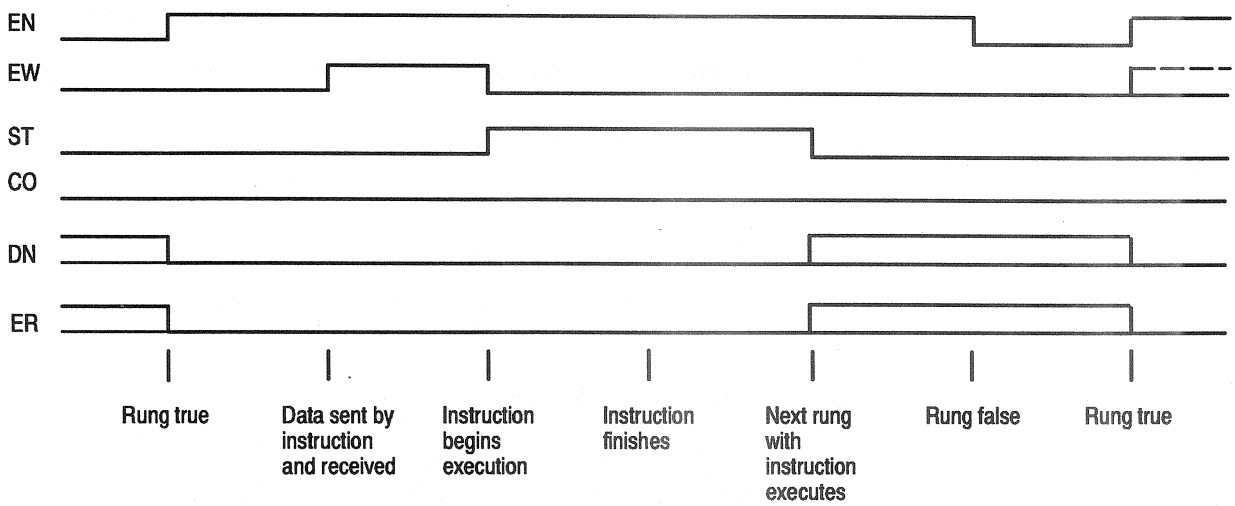
2. The processor then queues the message request. When the message request enters the queue, the processor sets the .EW bit.

3. When the processor starts to process the message request, the processor sets the .ST bit. The next time the processor receives network control, the processor transmits the message.

4. If no errors occur during transmission, the processor sets the .DN bit after the last packet in the first execution of the MSG instruction transfers. If an error occurs, the processor sets the .ER bit and stores an error code in the lower byte of word 0 of the control block for PLC-5/10, -5/12, -5/15, -5/25 and word 1 of the control block for PLC-5/30, -5/40 and -5/60.

5. The next time the rung goes false, the processor resets the .EN bit. Then when the associated rung goes true again, the message transfer cycle starts again.

Figure 16.6
Timing Diagram for Status Bits in Non-Continuous MSG Instructions



MSG Timing

The time required for one PLC-5 processor to send or receive a message to/from another processor on the DH+ depends on the number of:

- stations on the DH+
- messages transmitted from active stations
- bytes of data of all transmitted messages
- message requests already in the queue

Timing starts with setting the enable bit and ends with setting the done bit in the ladder program of the station initiating the message. The order of operation is shown in Table 16.A:

Table 16.A
Message Instruction Operation

Receiving MSG (Station A reading/receiving from Station B)	Sending MSG (Station A writing/sending to Station B)
station A enables the message instruction in the ladder program	station A enables the message instruction in the ladder program
station A obtains the token and transmits the read command (station B acknowledges immediately)	station A obtains the token and transmits data (station B acknowledges immediately)
station B obtains the token and transmits requested data	station B stores the data in memory
station A receives the data and acknowledges immediately	station B obtains the token to respond that the write is complete
station A sets the done bit	station A sets the done bit when it receives a response

You can estimate the time (in milliseconds) for transmitting one packet using the following formulas:

Processor Type:	Formula:
PLC-5/10, -5/12, -5/15, -5/25	Message time = TP + TT + OH + P + 8(number of messages)
PLC-5/30, -5/40, -5/60	Message time = TP + TT + OH + 8(number of messages)

where:

TP = token pass = (1.5)(1 + number of stations on DH+)

TT = transmission time = (0.28)(number of data words)
Number of data words is all transmitted messages for one token pass around the DH+.

OH = DH+ overhead = 20ms

P = longest program scan for any processor on the DH+ (application value in milliseconds)

Error Codes

When the processor detects an error during the transfer of message data, the processor sets the .ER bit and enters an error code that you can monitor from your programming terminal. If the message is non-continuous, the processor sets the .ER bit the first time the processor scans the MSG instruction.

Table 16.B
Errors Detected By the Processor

Code:		Description (Displayed on the Data Monitor screen):
PLC-5/10, -5/12, -5/15, -5/25 decimal - low byte of word 0 of the control block	**PLC-5/30, -5/40, -5/60** hexadecimal - word 1 of the control block	
55	0037	message timed out in local processor
129	1000	illegal command from local processor
130	2000	communication module not working
131	0083	processor is disconnected
132	4000	processor connected but faulted (hardware)
133	5000	you used the wrong station number
134	6000	requested function is not available
135	7000	processor is in program mode
136	8000	processor's compatibility file does not exist
137	0089	processor's message buffer is full
139	B000	processor is downloading so it is inaccessible
146	0092	no response (regardless of station type)
211	00D3	you formatted the control block incorrectly
213	00D5	incorrect address for the local data table
231	F001	processor incorrectly converted the address
232	F002	incomplete address
233	F003	incorrect address
236	F006	addressed file does not exist in target processor
237	F007	destination file is too small for number of words requested
240	F00A	target processor cannot put requested information in packets
241	F00B	privilege error, access denied
242	F00C	requested function is not available
243	F00D	request is redundant
247	F011	data type requested does not match data available
248	F012	incorrect command parameters

Chapter 17

ASCII Instructions ABL, ACB, ACI, ACN, AEX, AIC, AHL, ARD, ARL, ASC, ASR, AWA, AWT

PLC-5/30, -5/40, -5/40L, -5/60 and -5/60L Processors only

Using ASCII Instructions

The ASCII instructions let you read, write, compare and convert ASCII strings. These instructions are only supported by PLC-5/30, -5/40 and -5/60 processors. Table 17.A lists the available ASCII instructions.

Table 17.A
Available ASCII Instructions

If You Want to:	Use this Instruction:	Found on Page:
See how many characters are in the buffer, up to and including the end of line character	ABL *	17-3
See how many total characters are in the buffer	ACB *	17-4
Convert a string to an integer value	ACI *	17-5
Concatenate two strings into one	ACN *	17-5
Extract a portion of a string to create a new string	AEX *	17-6
Configure your modem handshake lines	AHL *	17-7
Convert an integer value to a string	AIC *	17-8
Read characters from the buffer and put into a string	ARD *	17-8
Read one line of characters from the buffer and put into a string	ARL *	17-10
Search a string for another string	ASC *	17-12
Compare two strings	ASR *	17-13
Write a string with user-configured characters appended	AWA *	17-13
Write a string	AWT *	17-15

* These instructions are only supported by PLC-5/30, -5/40 and -5/60 processors.

There are two types of ASCII instructions:

- ASCII port control – read, write, set/reset handshake lines, examine the length of the buffer (ARD, ARL, AWT, AWA, AHL, ACB, ABL)
- ASCII string – manipulate string data, such as compare, search, extract, concatenate, convert to/from integer (ASR, ASC, AEX, ACN, ACI, AIC)

Using Status Bits

You can examine status bits in the ladder program to examine some event. The processor changes the states of status bits as the processor executes the instruction. You address the status bits by mnemonic (or bit number) in the control element address.

The ASCII instructions use the length (.LEN) and position (.POS) fields in some instructions as well as the following status bits:

Bit:	Description:
08	Found (.FD)
10	Unload (.UL)
11	Error (.ER)
12	Synchronous Done (.EM)
13	Asynchronous Done (.DN)
14	Queue (.EU)
15	Enable (.EN)

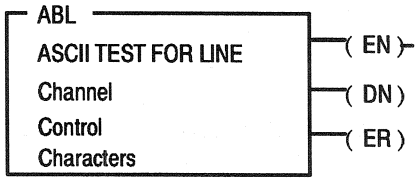
Using Strings

You can address string lengths by adding a .LEN to any string address (for example, ST17:1.LEN).

String lengths must be between 0 and 82. In general, lengths that are outside of this range causes the processor to set a minor fault (S:17/8) and the instruction is not executed. See the individual instruction for more details.

Important: You configure the append or end-of-line characters on the Channel Configuration screen. The default append characters are carriage return and line feed; the default end-of-line (termination) character is carriage return. For more information, see the channel configuration chapter in the "Installation and Configuration" module.

Test Buffer for Line (ABL) (PLC-5/30, -5/40, -5/60 only)

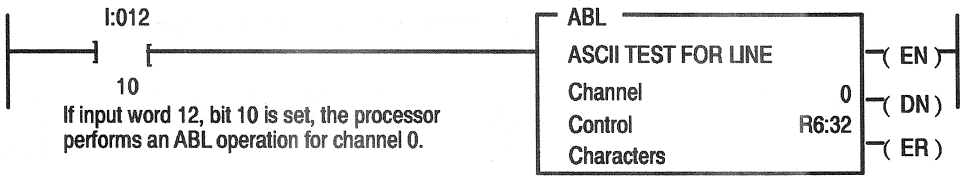


Description:
Use the ABL instruction to see how many characters are in the buffer, up to and including the end-of-line characters (termination). On a false-to-true transition, the system reports the number of characters in the Position field, and sets the done bit. The serial port must be in User mode.

Entering Parameters:
To use the ABL instruction, you must supply the following information:

- **Channel** is the number of the RS-232 port (for example 0).
- **Control** is the address of a control file element used for the control status bits.
- **Characters** is the number of characters in the buffer (including the end-of-line/termination characters) that the processor finds (0–256). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

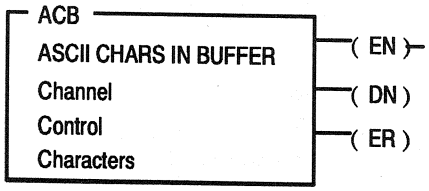
The processor determines the number of characters (up to and including the end-of-line/termination characters) and puts this value in the position field. The done bit is then set. If a zero appears in the position field, no end-of-line/termination characters were found. The .FD bit is set if the position field was set to a non-zero value.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

The error bit (.ER) is set during the execution of the instruction if:

- the instruction is aborted – serial port not in User mode
- the instruction is aborted due to processor mode change

Number of Characters in Buffer (ACB) (PLC-5/30, -5/40, -5/60 only)

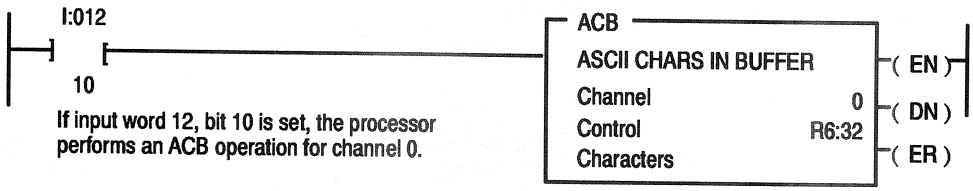


Description:
Use the ACB instruction to see how many total characters are in the buffer. On a false-to-true transition, the system determines the total number of characters and reports it in the Characters field. The serial port must be in User mode.

Entering Parameters:
To use the ACB instruction, you must provide the processor the following information:

- **Channel** is the number of the RS-232 port (for example, 0).
- **Control** is the address of a control file element used for control status bits.
- **Characters** is the number of the characters in the buffer that the processor finds (0–256). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. When the instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

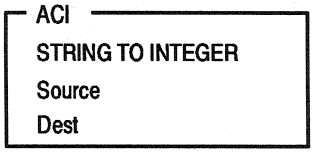
The processor determines the number of characters in the buffer and puts this value in the position field. The done bit is then set. If a zero appears in the position field, no characters were found. The .FD bit is set if the position field was set to a non-zero value.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

The error bit (.ER) is set during the execution of the instruction if:

- the instruction is aborted – serial port not in User mode
- the instruction is aborted due to processor mode change

ASCII String to Integer (ACI) (PLC-5/30, -5/40, -5/60 only)



Description:
Use the ACI instruction to convert an ASCII string to an integer value between -32768 and 32767.

The processor searches a source string element (file type ST) for the first character that is between 0 and 9. All numeric characters are extracted until a non-numeric character or the end of the string is reached. Commas and signs (–, +) are allowed in the string.

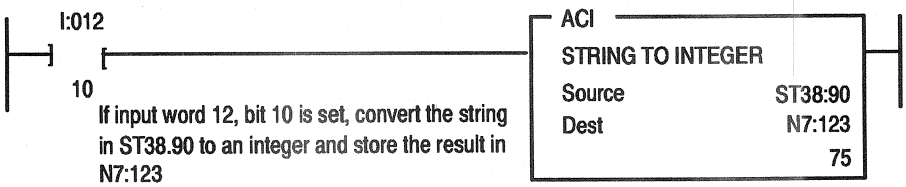
The extracted numeric string is then converted to an integer between -32768 and 32767.

If no numeric characters are found, no action is taken. Also, if the string has an invalid length (less than zero or greater than 82), the fault bit is set (S:17/8) and the instruction is not executed.

This instruction also sets the arithmetic flagss (found in word 0, bits 0–3 in the processor status file S):

Bit:	Description:	Indicating:
S:0/0	Carry (C)	the carry was generated while converting the string to an integer
S:0/1	Overflow (V)	the integer value was outside of the valid range
S:0/2	Zero (Z)	the integer value is zero
S:0/3	Sign (S)	the integer value is negative

Example:

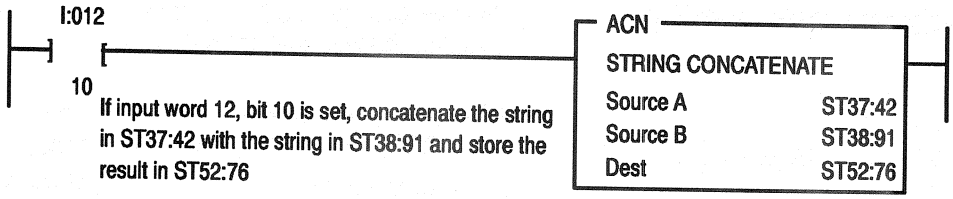


ASCII String Concatenate (ACN) (PLC-5/30, -5/40, -5/60 only)

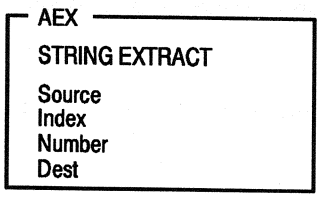
Description:
The ACN instruction combines two strings together using ASCII strings as operands. The second string is appended to the first and the result stored in the destination.

The result is limited to 82 characters, so if the result is longer than 82 characters, only the first 82 are written to the destination file and the error bit (S:17/8) is set. Also, if the length of either string is invalid (less than zero or greater than 82), the fault bit is set and the string at the destination address is not changed.

Example:



ASCII String Extract (AEX) (PLC-5/30, -5/40, -5/60 only)

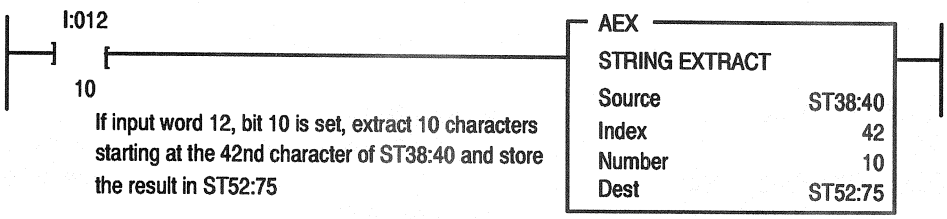


Description:
Use the AEX instruction to create a new string by taking a portion of an existing string.

Entering Parameters:
To use the AEX instruction, you must provide the processor the following information:

- **Source** is the existing string.
- **Index** is the starting position (from 1 to 82) of the portion of the string you want to extract. (An index of 1 indicates the left-most character of the string.)
- **Number** is the number of characters (from 0 to 82) you want to extract, starting at the indexed position. If the Index plus the Number is greater than the total characters in the source string, the destination string will be the characters from the index to the end of the source string. If you enter 0 for the number, the destination string length is set to zero.
- **Destination** is the string element (ST) where you want the extracted string stored.

Example:

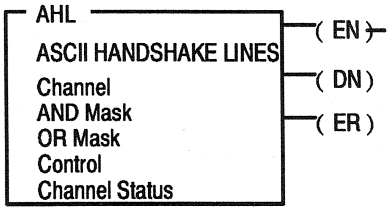


The following conditions cause the processor to set the fault bit (S:17/8):

- invalid string length or string length of zero
- index or number values outside of range
- index value greater than the length of the source string

The destination string will not change in any of the above instances.

ASCII Set or Reset Handshake Lines (AHL) (PLC-5/30, -5/40, -5/60 only)



Description:
Use the AHL instruction to set or reset the RS-232 DTR and RTS handshake control lines for your modem. On a false-to-true transition, the system uses the two masks to determine whether to set or reset the DTR and RTS lines, or leave them unchanged.

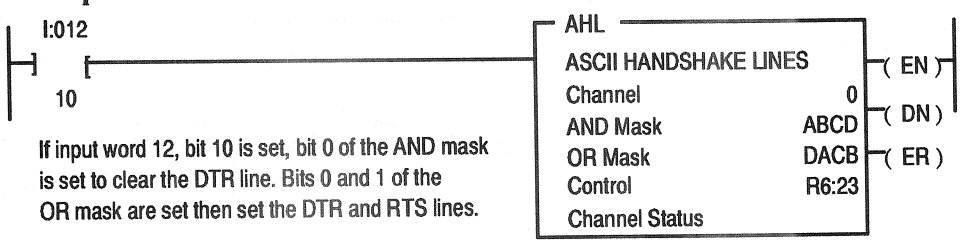
Important: Before you use this instruction make sure that you do not conflict with the automatic control lines on your modem.

Entering Parameters:
To use the AHL instruction, you must provide the following information:

- **Channel** is the number of the RS-232 port that you want to use. Currently, only channel 0 can be set or reset.
- **AND Mask** is the mask to reset the DTR and RTS control lines. Bit 0 corresponds to the DTR line and bit 1 corresponds to the RTS line. A 1 at the mask bit causes the line to be reset; a 0 leaves the line unchanged.
- **OR Mask** is the mask to set the DTR and RTS control lines. Bit 0 corresponds to the DTR line and bit 1 corresponds to the RTS line. A 1 at the mask bit causes the line to be set; a 0 leaves the line unchanged.
- **Control** is the address of the result control structure in the control area of memory for the result.
- **Channel Status** displays the current status (0000 to FFFF) of the handshake lines for the channel specified above. This field is display only; convert the hexadecimal status to binary and refer to the table below:

Bit	15 ---- 6	4	3	2	1	0
Line	unused	DTR	DCD	DSR	RTS	CTS

Example:



If input word 12, bit 10 is set, bit 0 of the AND mask is set to clear the DTR line. Bits 0 and 1 of the OR mask are set then set the DTR and RTS lines.

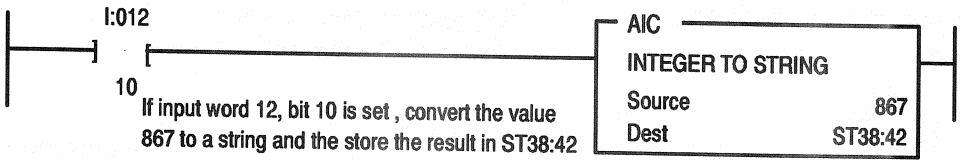
The error bit (.ER) is set during the execution of the instruction if:

- the instruction is aborted due to processor mode change

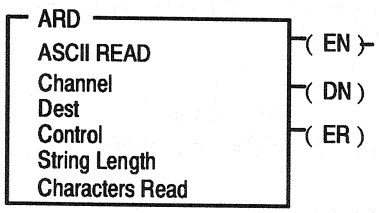
ASCII Integer to String (AIC) (PLC-5/30, -5/40, -5/60 only)

Description:
Use the AIC instruction converts an integer value (between -32768 and 32767) to an ASCII string. The source can be a constant or an integer address.

Example:



ASCII Read Characters (ARD) (PLC-5/30, -5/40, -5/60 only)

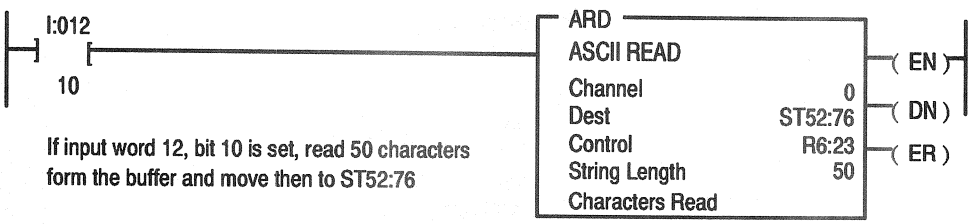


Description:
Use the ARD instruction to read characters from the buffer and store them in a string. To repeat the operation, the rung must go from false to true. The serial port must be in User mode.

Entering Parameters:
To use the ARD instruction, provide the following information:

- **Channel** is the number of the RS-232 port (for example, 0).
- **Control** is the control file element used for the control status bits.
- **Destination** is the string element where you want the characters stored.
- **String Length** is the number of characters you want to read from the buffer. The maximum is 82 characters. If you specify a length larger than 82, only 82 characters will be read. (If you specify 0, the string length defaults to 82.)
- **Characters Read** is the number of characters that the processor moved from the buffer to the string (0 to 82). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Once the requested number of characters are in the buffer, the characters are moved to the destination string. The number of characters moved is put in the position word of the control element and the done bit is set.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

You can use the .UL bit to terminate an ARD instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port is not sending data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

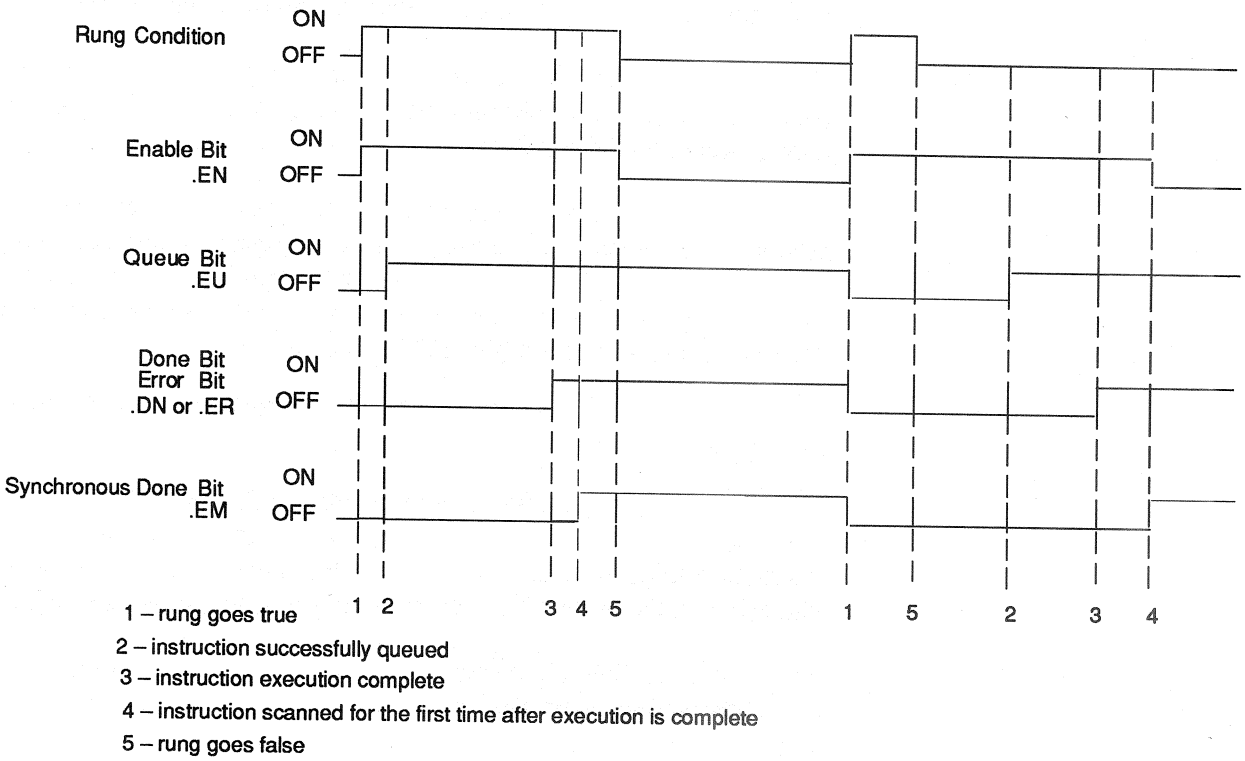
Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an ARD instruction starts executing with the .UL bit already set and there are **no** characters in the buffer, the instruction terminates. If an ARD instruction starts executing with the .UL bit already set and there **are** characters in the buffer, the instruction proceeds to normal completion.

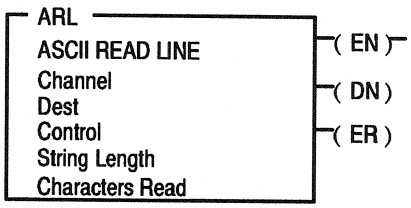
The error bit (.ER) is set during the execution of the instruction if:

- the instruction is aborted – serial port not in User mode
- the instruction is aborted due to processor mode change
- when using a modem, the modem is disconnected

Figure 17.1
Example ARD Timing Diagram



ASCII Read Line (ARL) (PLC-5/30, -5/40, -5/60 only)



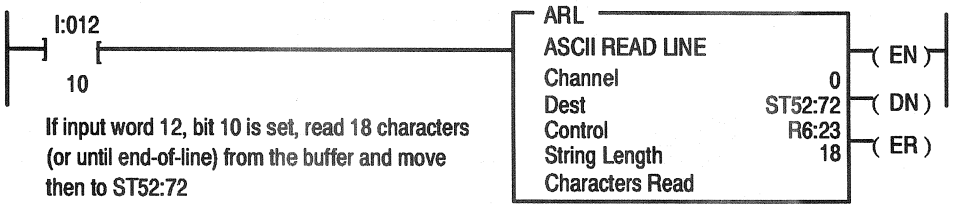
Description:
Use the ARL instruction to read characters from the buffer up to and including the end-of-line (termination) characters and store them in a string. The end-of-line characters are specified on the Channel Configuration screen (the default is a carriage return). For more information on channel configuration, see the chapter in the "Configuration and Maintenance" module. The serial port must be in User mode.

Entering Parameters:
To use the ARL instruction, you must provide the following information:

- **Channel** is the number of the RS-232 port (for example, 0).
- **Control** is the address of the control file element used for the control status bits.
- **Destination** is the string element where you want the string stored.

- **String Length** is the number of characters (maximum 82) you want to read from the buffer. If the processor finds the end-of-line characters before reading the number of characters you specified, only those characters read and the end-of-line are moved to the destination.
- **Characters Read** is the number of characters that the processor moved from the buffer to the string (0 to 82). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Once the requested number of characters (or the end-of-line characters) are in the buffer, all characters (including the end-of-line characters) are moved to the destination string. The number of characters moved is stored in the position word of the control element and the done bit is set.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

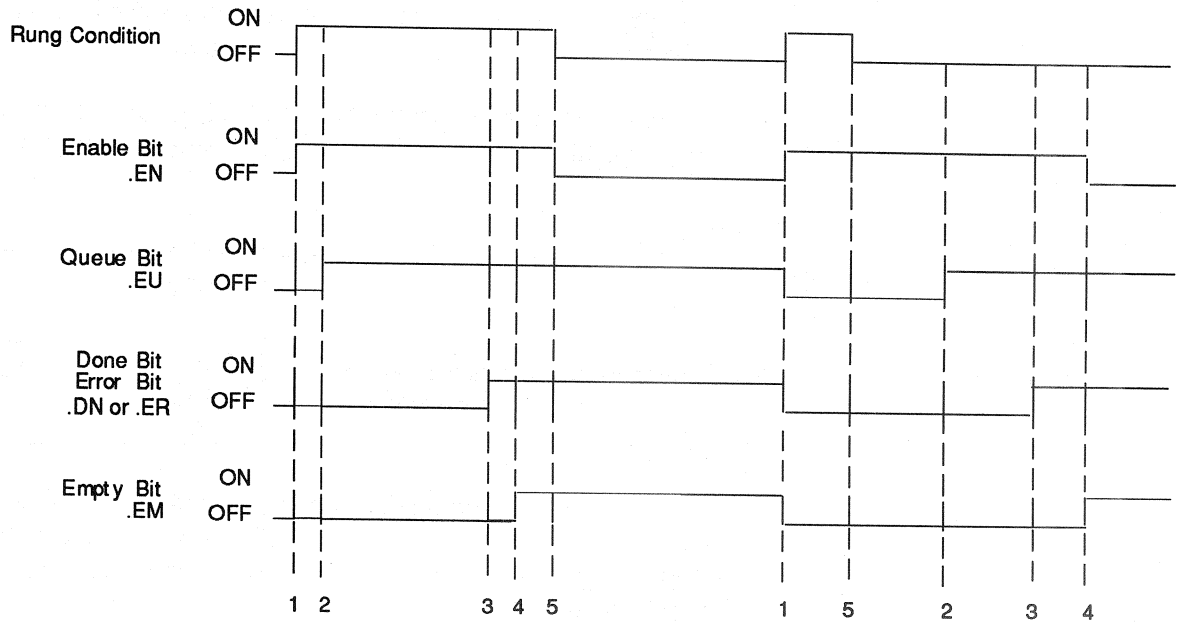
You can use the .UL bit to terminate an ARL instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port is not sending data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an ARL instruction starts executing with the .UL bit already set and there are **no** characters in the buffer, the instruction terminates. If an ARL instruction starts executing with the .UL bit already set and there **are** characters in the buffer, the instruction proceeds to normal completion.

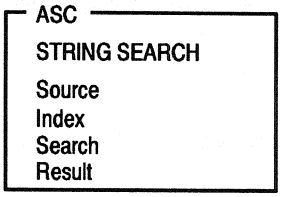
The error bit (.ER) is set during the execution of the instruction if the channel is in system mode (or switches to system mode), the processor switches to program/test mode or if the modem is lost (when using modem control).

Figure 17.2
Example ARL Timing Diagram



1 – rung goes true
2 – instruction successfully queued
3 – instruction execution complete
4 – instruction scanned for the first time after execution is complete
5 – rung goes false

ASCII String Search (ASC) (PLC-5/30, -5/40, -5/60 only)

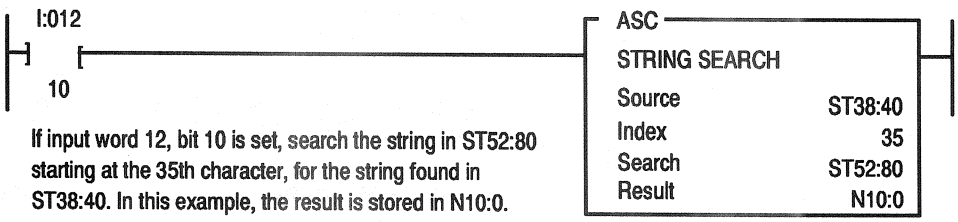


Description:
Use the ASC instruction to search an existing string (search string) for an occurrence of the source string.

Entering Parameters:
To use the ASC instruction, you must provide the processor the following information:

- **Search** is the string you want to examine.
- **Source** is the string you want to find when examining the search string.
- **Index** is the starting position (from 1 to 82) of the portion of the search string you want to search. An index of 1 indicates the left-most character.
- **Result** is an integer file where the processor stores the position of the search string where the source string begins. If no match is found, 0 is stored in the result.

Example:



If input word 12, bit 10 is set, search the string in ST52:80 starting at the 35th character, for the string found in ST38:40. In this example, the result is stored in N10:0.

The following conditions cause the processor to set the fault bit (S:17/8):

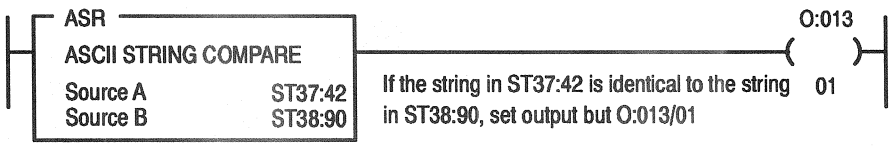
- invalid string length or string length of zero
- index values outside of range
- index value greater than the length of the source string

The result is set to zero in any of the above instances.

ASCII String Compare (ASR) (PLC-5/30, -5/40, -5/60 only)

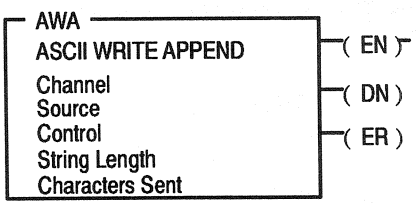
Description:
Use the ASR instruction to compare two ASCII strings. The system looks for a match in length and upper/lower case. If the two strings are identical, the rung is true; if there are any differences, the rung is false.

Example:



An invalid string length causes the processor to set the fault bit (S:17/8), and the rung is false.

ASCII Write with Append (AWA) (PLC-5/30, -5/40, -5/60 only)



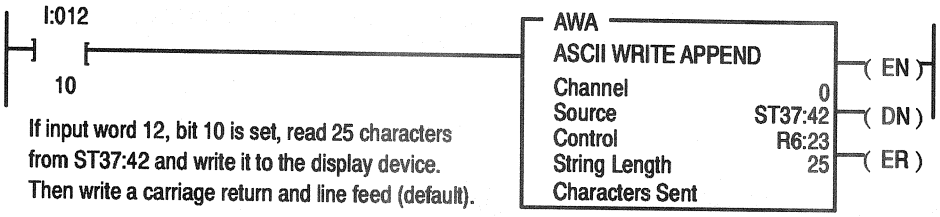
Description:
Use the AWA instruction to write characters from a source string to some display device. This append instruction adds 1 or 2 characters (which you configure in the channel configuration table). The default is a carriage return and line feed appended to the end of the string. You can use this instruction with the serial port in User or System mode.

Entering Parameters:
To use the AWA instruction, you must provide the following information:

- **Channel** is the number of the RS-232 port (for example, 0).
- **Source** is the string you want to write.

- **Control** is the address of the control file element used for the control status bits.
- **String Length** is the maximum number of characters you want to write from the source string (0 to 82). If you enter 0, the entire string will be written.
- **Characters Sent** is the number of characters that the processor sent to the display area (0 to 82). Only after the entire string is sent is this field updated (no running total for each character sent is stored). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Twenty-five characters form the start of string ST37:42 are sent to the display device and then user-configured append characters are sent. The done bit is set and a value of 27 is sent to the position word.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit to act as a secondary done bit corresponding to the program scan.

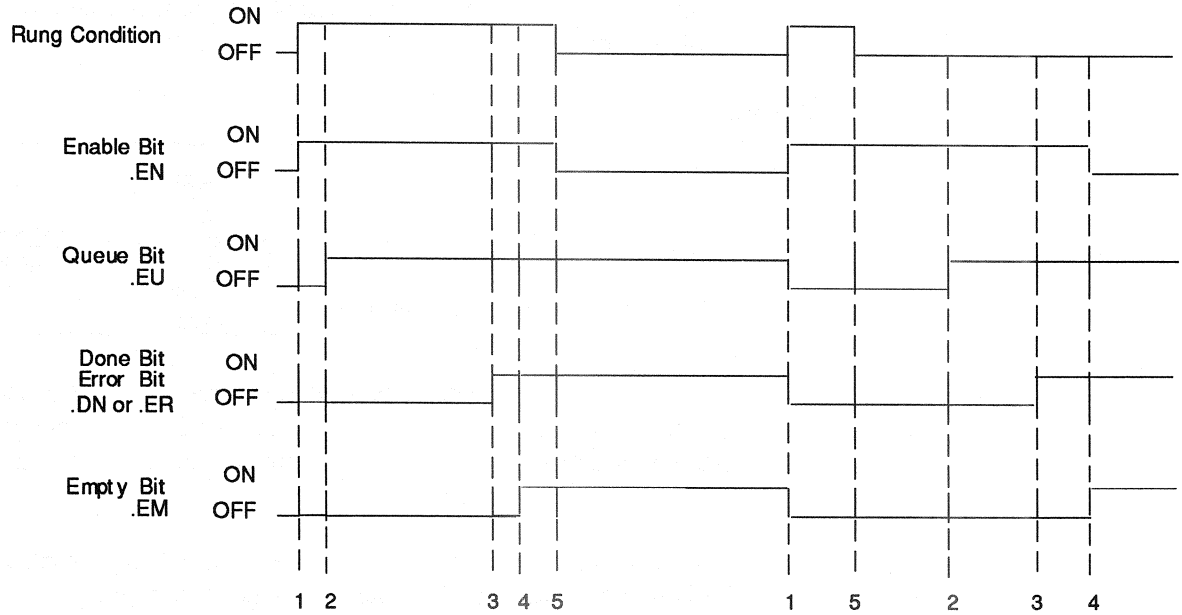
You can use the .UL bit to terminate an AWA instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port cannot accept data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an AWA instruction starts executing with the .UL bit already set, the instruction aborts immediately.

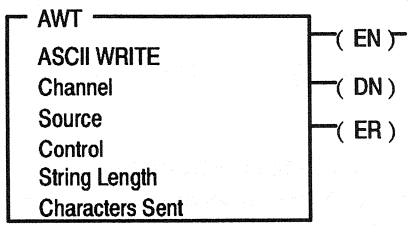
The error bit (.ER) is set during the execution of the instruction if the instruction is aborted due to processor mode change or if the modem becomes lost (when using modem control). If the modem was already lost, the instruction still executes.

Figure 17.3
Example AWA Timing Diagram



1 – rung goes true
2 – instruction successfully queued
3 – instruction execution complete
4 – instruction scanned for the first time after execution is complete
5 – rung goes false

ASCII Write (AWT) (PLC-5/30, -5/40, -5/60 only)



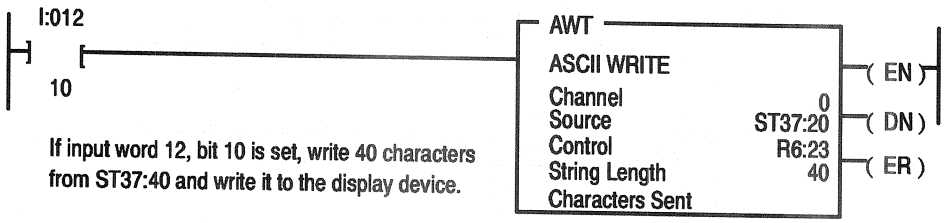
Description:
Use the AWT instruction to write characters from a source string to some display device. To repeat the instruction, the rung must go from false to true. You can use this instruction with the port in System or User mode.

Entering Parameters:
To use the AWT instruction, you must provide the following information:

- **Channel** is the number of the RS-232 port (for example, 0).
- **Source** is the string you want to write.
- **Control** is the address of the control file element used for the control status file.
- **String Length** is the maximum number of characters you want to write from the source string (0 to 82). If you enter 0, the entire string will be written.

- **Characters Sent** is the number of characters that the processor sent to the display area (0 to 82). Only after the entire string is sent is this field updated (no running total for each character sent is stored). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Forty characters from string ST37:40 are sent through channel 0. The done bit is set and a value of 40 is sent to the position word.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

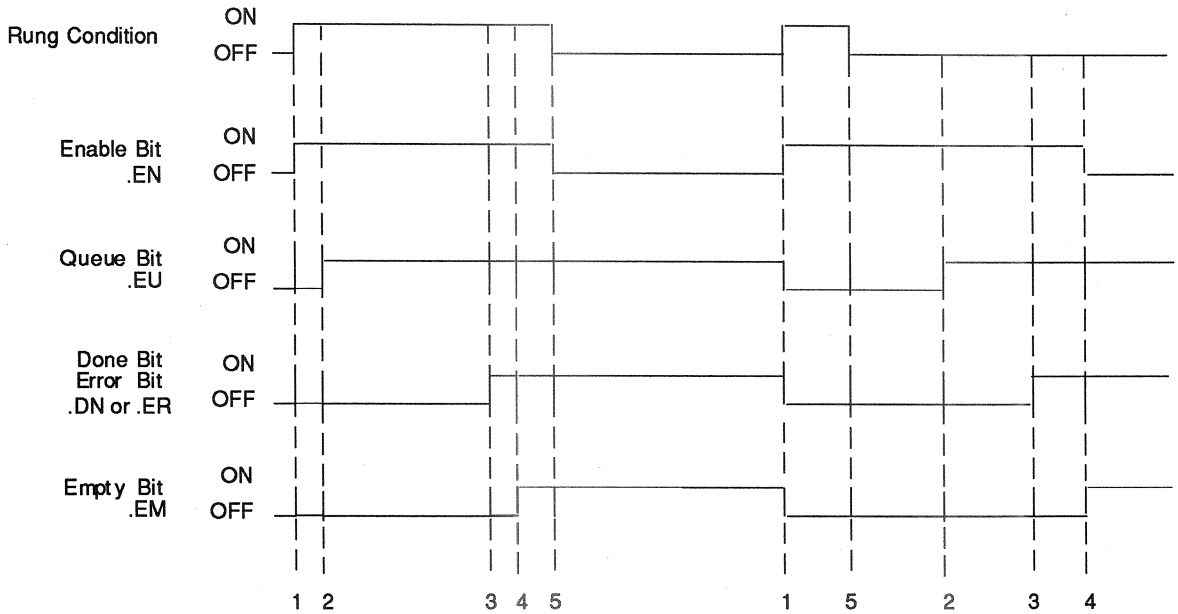
You can use the .UL bit to terminate an AWT instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port cannot accept data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an AWT instruction starts executing with the .UL bit already set, the instruction aborts immediately.

The error bit (.ER) is set during the execution of the construction if the processor switches to program or test mode or if the modem becomes lost (when using modem control). If the modem was already lost, the instruction still executes.

Figure 17.4
Example AWT Timing Diagram



1 – rung goes true
2 – instruction successfully queued
3 – instruction execution complete
4 – instruction scanned for the first time after execution is complete
5 – rung goes false

Chapter 18

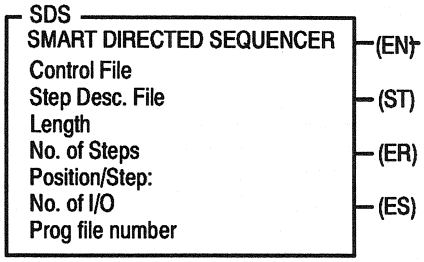
Custom Application Routine Instructions SDS, DFA

Chapter Objectives

This chapter introduces the Custom Application Routine (CAR) instructions and describes procedures for entering them from 6200 Series Software. The CAR instructions that are available: SDS instruction, DFA instruction. You need the Custom Application Routine (CAR) software in order to use these instructions (software catalog number 6401-, 6403-, 6411-, 6413-DDMC).

For more information on the SDS or DFA CAR utilities, see the Distributed Diagnostic and Machine Control User Manual (publication 6401-6.5.1).

Smart Directed Sequencer Overview



The Smart Directed Sequencer (SDS) instruction resides in ladder logic and provides state control that can be used to characterize normal and abnormal conditions.

Currently, the SDS instruction allows two basic types of logic equations:

- Transitional
- Combinatorial

Transitional Equations

The SDS instruction which uses transition equations provides traditional state-based control. Essentially, this type of SDS instruction is built around the state transition concept, where each input transition directs the instruction to a unique next state using a logical OR structure. In other words, one input change directs the instruction to step A, another to step B, etc..

Combinatorial Equations

The SDS instruction which uses combinatorial equations allows for the ANDing of inputs in addition to the OR function used in transition equations. This allows complex combinations to be accomodated more easily within the SDS framework with a minimum number of steps.

For more information on these equations, see the Distributed Diagnostic and Machine Control User Manual (publication 6401-6.5.1).

Programming the SDS Instruction

Once you have determined where you want to apply the SDS instruction, you can enter it into your ladder program and configure the instruction. You perform the following tasks to program the SDS instruction:

- download the SDS CAR
- enter the SDS instruction
- enter the configuration information
- enter I/O information

To enter the SDS instruction into your ladder program, do the following:

Download the SDS CAR utility

To download the the SDS CAR utility into your program, do the following:

To get to the Program Directory screen:

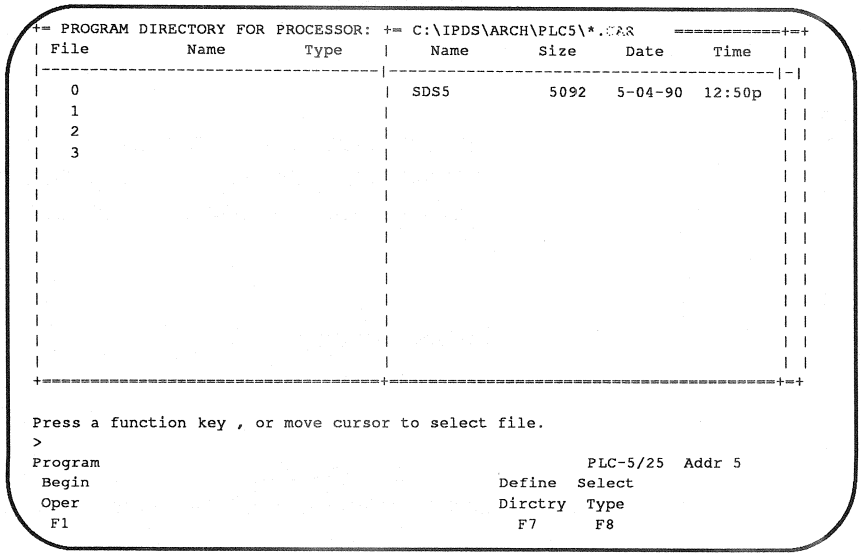
6200 Series Software Main Menu
[F1] - Online Programming
or
[F3] - Offline Programming

1. Start on the Program Directory screen.
2. Press `[F2] - Save Restore`

   The system displays the Save Restore function keys.
3. Press `[F4] - Restore Program`

   The system displays a window containing a file directory of a specific type (Figure 18.5).

Figure 18.5
File Directory Window (PLC-5)



4. Press `[F8] - Select Type` until the CAR directory appears in the file directory window.

5. Cursor to the desired CAR file.

6. Enter the CAR file:

   a. Press `[F1] - Begin Restore`

      The system prompts you to enter a file number.

   b. Type the file number and press `[ENTER]`

      The system returns to the Program Directory.

Enter or Modify an SDS Instruction

We have included entering and modifying SDS instructions in one section because these procedures are so similar. You use the same screens to perform these operations with only slightly different keystrokes.

To enter or modify an SDS instruction in your ladder program, do the following:

1. Cursor to the program you want to edit in the Program Directory and press `[F8] - Monitor File`

   The system displays your ladder program and the Ladder Editor main menu.

2. Cursor to the rung where you want to position the SDS instruction.

   If you are modifying an existing rung, cursor to the SDS rung.

   Important: When cursored on the R rung of an IR rung pair, the SDS configuration utility uses the configuration information from the I rung.

3. Press `[F10] - Edit`

   The system displays the Ladder Editing screen.

4. If you are editing an existing rung, press `[F5] - Modify Rung`

   If you are entering a new rung, press `[F4] - Insert Rung`

   The system displays the Edit Rung screen.

5. If you want to change an existing instruction, press **[F5] - Modify Instruction**

   If you are entering a new instruction, press **[F4] - Insert Instruction**

   The system prompts you for the instruction you want to add.

6. Type **SDS,** if it does not already appear by the prompt, and press **[ENTER],** entering the instruction. (If you are modifying an existing instruction, **SDS** already appears at the prompt.)

   As an alternate method of entering the instruction, you may:

   - Press **[F10] - Others**
   - Press **[F9] - Seqncer** from the Instruction Classes function keys.
   - Press **[F6] - SDS** from the Shift Sequencer function keys.

   The system displays the SDS instruction (Figure 18.6).

Figure 18.6
SDS Instruction (PLC-5)

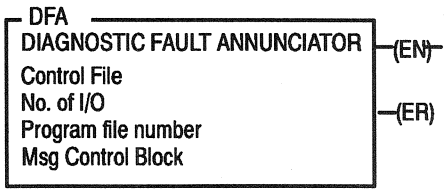
```
|                                        +SDS-------------------------+       |
+----------------------------------------+SMART DIRECTED SEQUENCER     +-(EN)-+
|                                        |Control File                |       |
|                                        |Step Desc. File             +-(ST)  |
|                                        |Length                      |       |
|                                        |No. of Steps                +-(ER)  |
|                                        |Position/Step:              |       |
|                                        |No. of I/O                  +-(ES)  |
|                                        |Program File No.            |       |
|                                        +----------------------------+       |
|                                                                             |
+-------------------------------[END OF FILE]-------------------------------+
|                                                                             |
```

```
Enter the operand.
Enter                  the                  Control                  File                  address>
 Program    Forces:Disabled  Edits:None                  PLC-5/25 Addr 5
```

Important: Note that you cannot use the PLC-5/30, -5/40 and -5/60 processor data types (BT, PD, MG, ST, SC) within the I/O list of the SDS instruction; the instruction will not operate properly is you use one of these data types.

For more information, see the Distributed Diagnostic and Machine Control User Manual (publication 6401-6.5.1).

Diagnostic Fault Annunciator Overview



The Diagnostic Fault Annunciator (DFA) instruction is a monitoring only instruction; that is, it monitors inputs you define, but it cannot control outputs. Valid inputs can be:

- storage points, such as binary bits
- counter / timer done bits
- outputs (real or logical)
- any valid bit address
- lube level indicators
- alarms
- fault bits set by another device (like an IMC motion controller) or by ladder logic

If you currently have diagnostics programmed in ladder logic, you can use the DFA instruction to generate messages when a fault occurs. In addition, you can create other types of operational and diagnostic messages with the DFA instruction, such as tool change messages and operating instructions.

Programming the DFA Instruction

Once you have determined where you want to apply the DFA instruction, you can enter it into your ladder program and configure the instruction. You perform the following tasks to program the DFA instruction:

- download the DFA CAR
- enter the DFA instruction
- enter the configuration information
- enter I/O information

To download the the SDS CAR utility into your program, do the following:

Download the DFA CAR utility

To download the DFA CAR utility into your program:

To get to the Program Directory screen:

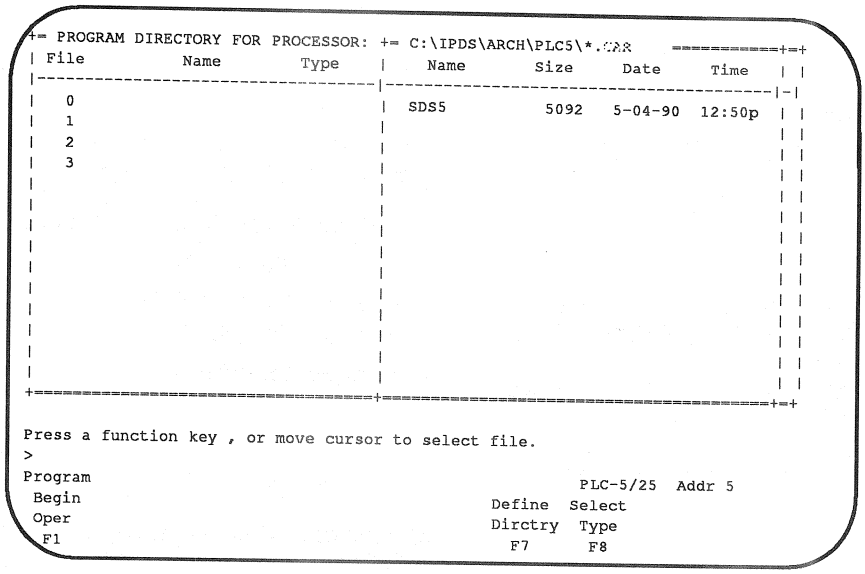
6200 Series Software Main Menu
[F1] - Online Programming
or
[F3] - Offline Programming

1. Start on the Program Directory screen.
2. Press `[F2] - Save Restore`

   The system displays the Save Restore function keys.
3. Press `[F4] - Restore Program`

   The system displays a window containing a file directory of a specific type (Figure 18.7).

Figure 18.7
File Directory Window



4. Press **[F8] – Select Type** until the CAR directory appears in the file directory window.

5. Cursor to the desired CAR file.

6. Enter the CAR file:

   a. Press **[F1] – Begin Restore**

      The system prompts you to enter a file number.

   b. Type the file number and press **[ENTER]**

      The system returns to the Program Directory.

Enter or Modify a DFA Instruction

We have included entering and modifying DFA instructions in one section because these procedures are so similar. You use the same screens to perform these operations with only slightly different keystrokes.

To get to the Program Directory screen:

6200 Series Software Main Menu
[F1] - Online Programming
or
[F3] - Offline Programming

To enter or modify a DFA instruction in your ladder program, do the following:

1. Start on the Program Directory screen.

2. Cursor to the file you want to edit press **[F8] – Monitor File**

   The system displays your ladder program and the Ladder Editor main menu.

3. Cursor to the rung where you want to position the DFA instruction.

   If you are modifying an existing rung, cursor to the DFA rung.

4. Press **[F10] - Edit**

   The system displays the Ladder Editing screen.

5. If you are editing an existing rung, press **[F5] - Modify Rung**

   If you are entering a new rung, press **[F4] - Insert Rung**

   The system displays the Edit Rung screen.

6. If you want to change an existing instruction, press **[F5] - Modify Instruction**

   If you are entering a new instruction, press **[F4] - Insert Instruction**

   The system prompts you for the instruction you want to add.

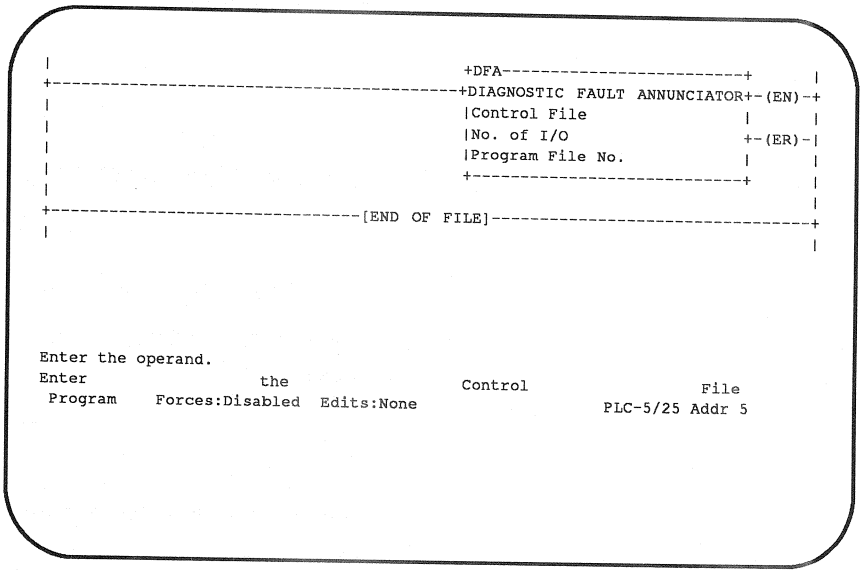
7. Type **DFA**, and press **[ENTER]**, entering the instruction. (If you are modifying an existing instruction, DFA already appears at the prompt.

   As an alternate method of entering the instruction, you may:

   - Press **[F10] - Others**
   - Press **[F9] - Seqncer** from the Instruction Classes function keys.
   - Press **[F5] - DFA** from the Sequencer function keys.

   The system displays the DFA instruction (Figure 18.8).

Figure 18.8
DFA Instruction



For more information on the SDS or DFA CAR utilities, see the Distributed Diagnostic and Machine Control User Manual (publication 6401-6.5.1).

Appendix A

Instruction Timing and Memory Requirements

Instruction Timing and Memory Requirements

The time it takes for a processor to scan an instruction depends on the type of instruction, the type of addressing, the type of data, whether the instruction has to convert data, and whether the instruction is true or false.

The timing and memory requirements estimates in this chapter have the following assumptions:

- direct addressing
- integer data, except where noted
- no data-type conversions
- addresses within the first 4098 words of the data table
- execution times shown in µs

Memory requirements refer to the number of words the instruction uses. In some cases, an instruction may have a range of memory requirements. The range of words exists because the instruction can use different types of data.

Important: The tables are divided into instructions times and memory requirements that are specific to each processor. Refer to the table below:

If You Are Using This Processor:	See Page:
PLC-5/30 series A, PLC-5/40, -5/40L series A and B, PLC-5/60, -5/60L series A and B:	
Bit and Word Instructions	A-2
File Instructions	A-5
PLC-5/10, -5/12, -5/15, -5/25 (all series):	
Bit and Word Instructions	A-9
File Instructions	A-12

Timing for PLC-5/30, -5/40, -5/40L, -5/60 and -5/60L Processors

Bit and Word Instructions

Table A.A shows timing and memory requirements for bit and word instructions for PLC-5/30, -5/40, -5/40L, -5/60 and -5/60L processors.

Table A.A
Timing and Memory Requirements for Bit and Word Instructions for PLC-5/30, -5/40, -5/40L, -5/60 and -5/60L processors

			PLC-5/40, -5/40L, -5/60, -5/60L Series A processors				PLC-5/30 Series A and PLC-5/40, -5/40L, -5/60, -5/60L Series B processors			
Category	Code	Title	Execution Time (µs) integer		Execution Time (µs) floating point		Execution Time (µs) integer		Execution Time (µs) floating point	
			True	False	True	False	True	False	True	False
Relay	XIC	examine if closed	.32	.16			.32	.16		
	XIO	examine if open	.32	.16			.32	.16		
	OTL	output latch	.48	.16			.48	.16		
	OTU	output unlatch	.48	.16			.48	.16		
	OTE	output energize	.48	.48			.48	.48		
Branch		branch end	.16	.16			.16	.16		
		next branch								
		branch start								
Timer and Counter	TON	timer on (0.01 base)	3.8	2.6			3.8	2.6		
		(1.0 base)	4.1	2.5			4.1	2.5		
	TOF	timer off (0.01 base)	2.6	3.2			2.6	3.2		
		(1.0 base)	2.6	3.2			2.6	3.2		
	RTO	retentive timer on (0.01 base)	3.8	2.4			3.8	2.4		
		(1.0 base)	4.1	2.3			4.1	2.3		
	CTU	count up	3.4	3.4			3.4	3.4		
	CTD	count down	3.3	3.4			3.3	3.4		
	RES	reset	2.2	1.0			2.2	1.0		

			PLC-5/40, -5/40L, -5/60, -5/60L Series A processors				PLC-5/30 Series A and PLC-5/40, -5/40L, -5/60, -5/60L Series B processors			
Category	Code	Title	Execution Time (µs) integer		Execution Time (µs) floating point		Execution Time (µs) integer		Execution Time (µs) floating point	
			True	False	True	False	True	False	True	False
Arithmetic	ADD	add	6.1	1.4	344	1.4	6.1	1.4	14.9	1.4
	SUB	subtract	6.2	1.4	346	1.4	6.2	1.4	15.6	1.4
	MUL	multiply	256	1.4	346	1.4	9.9	1.4	18.2	1.4
	DIV	divides	12.2	1.4	346	1.4	12.2	1.4	23.4	1.4
	SQR	square root	9.9	1.3	313	1.3	9.9	1.3	35.6	1.3
	NEG	negate	4.8	1.3	250	1.3	4.8	1.3	6.0	1.3
	CLR	clear	3.4	1.1	229	1.1	3.4	1.1	3.9	1.1
	AVE	average file	470 +E92.2	261	467+E92.3	262	179+E25.8	54	184+E22.6	56
	STD	standard deviation	552+E175	261	552+E175	262	346+E84.1	84	357+E76.4	86
	TOD	convert to BCD	282	1.3			7.8	1.3		
	FRD	convert from BCD	265	1.3			8.1	1.3		
	RAD	radian	332	1.3	321	1.3	56.5	1.4	48.8	1.4
	DEG	degree	332	1.3	321	1.3	55.3	1.4	48.9	1.4
	SIN	sine			326	1.3			428	1.4
	COS	cosine			326	1.3			415	1.4
	TAN	tangent			333	1.3			519	1.4
	ASN	inverse sine			343	1.3			438	1.4
	ACS	inverse cosine			343	1.3			447	1.4
	ATN	inverse tangent			330	1.3			385	1.4
	LN	natural log	350	1.3	340	1.3	419	1.4	412	1.4
	LOG	log	352	1.3	343	1.3	420	1.4	413	1.4
	XPY	X to the power of Y	409	1.4	417	1.4	970	1.5	978	1.5
	SRT	sort file	302 + 25[E**1.34]	270	306 + 36[E**1.35]	270	219 + 22[E**1.34]	176	220 + 29[E**1.35]	176

E = number of elements acted on per scan

Note: SRT true is only an approximation. Actual execution time depends on the randomness of the numbers.

			PLC-5/40, -5/40L, -5/60, -5/60L Series A processors				PLC-5/30 Series A and PLC-5/40, -5/40L, -5/60, -5/60L Series B processors			
Category	Code	Title	Execution Time (µs) integer		Execution Time (µs) floating point		Execution Time (µs) integer		Execution Time (µs) floating point	
			True	False	True	False	True	False	True	False
Logic	AND	and	5.9	1.4			5.9	1.4		
	OR	or	5.9	1.4			5.9	1.4		
	XOR	exclusive or	5.9	1.4			5.9	1.4		
	NOT	not	4.6	1.3			4.6	1.3		
Move	MOV	move	4.5	1.3	246	1.3	4.5	1.3	5.6	1.3
	MVM	masked move	258	1.4			6.2	1.4		
	BTD	bit distributor	10.0	1.7			10.0	1.7		
Comparison	EQU	equal	3.8	1.0	4.3	1.0	3.8	1.0	4.6	1.0
	NEQ	not equal	3.8	1.0	4.2	1.0	3.8	1.0	4.5	1.0
	LES	less than	4.0	1.0	4.8	1.0	4.0	1.0	5.1	1.0
	LEQ	less than or equal	4.0	1.0	4.8	1.0	4.0	1.0	5.1	1.0
	GRT	greater than	4.0	1.0	4.8	1.0	4.0	1.0	5.1	1.0
	GEQ	greater than or equal	4.0	1.0	4.8	1.0	4.0	1.0	5.1	1.0
	LIM	limit test	6.1	1.1	7.9	1.1	6.1	1.1	8.4	1.1
	MEQ	mask compare if equal	5.1	1.1			5.1	1.1		
Compare	CMP	all	2.4 + (Σ [0.4 + I])	2.16 + WI[0.56]	2.4 + (Σ[0.4 + I])	2.16 + WI[0.56]	2.48 + (Σ[0.8 + I])	2.16 + WI[0.56]	2.48 + (Σ[0.8 + I])	2.16 + WI[0.56]
Compute	CPT	all	2.4 + (Σ [0.4 + I])	2.16 + WI[0.56]	2.4 + (Σ[0.4 + I])	2.16 + WI[0.56]	2.48 + (Σ[0.8 + I])	2.16 + WI[0.56]	2.48 + (Σ[0.8 + I])	2.16 + WI[0.5]

I = execution time of each instruction (operation) used within the CMP or CPT expression

WI = number of words of memory used by the instruction (operation) within the CMP or CPT expression

Note: CMP or CPT instructions are calculated with short direct addressing

File Instructions

Refer to Table A.B for the instruction timing for file instructions.

Table A.B
Timing and Memory Requirements for File Instructions for PLC-5/40, PLC-5/30, -5/40L, -5/60 and -5/60L

			PLC-5/40, -5/40L, -5/60, -5/60L Series A processors				PLC-5/30 Series A and PLC-5/40, -5/40L, -5/60, -5/60L Series B processors			
Category	**Code**	**Title**	**Time (µs) integer**		**Time (µs) floating point**		**Time (µs) integer**		**Time (µs) floating point**	
			True	**False**	**True**	**False**	**True**	**False**	**True**	**False**
File Arithmetic and Logic	FAL	all	11 + (Σ[2.0 + I])	6.16 + WI[0.16]	11 + (Σ[2.0 + I])	6.16 + WI[0.16]	11 + (Σ[2.3 + I])	6.16 + WI[0.16]	11 + (Σ[2.3 + I])	6.16 + WI[0.16]
File Search and Compare	FSC	all	11 + (Σ[2.0 + I])	6.16 + WI[0.16]	11 + (Σ[2.0 + I])	6.16 + WI[0.16]	11 + (Σ[2.3 + I])	6.16 + WI[0.16]	11 + (Σ[2.3 + I])	6.16 + WI[0.16]
File	COP	copy	15.6 + E[0.72]	1.4	17.1+E[1.44]	1.4	10.3+E[0.72]	1.4	11.2+ E[1.44]	1.4
		counter, timer, and control	16.2 + E[2.16]	1.4			11.7+ E[2.16]	1.4		
	FLL	fill	15.2 + E[0.64]	1.5	17.4+E[0.80]	1.5	11.2 +E[0.64]	1.5	13.0+W[0.80]	1.5
		counter, timer, and control	15.8 + E[1.60]	1.5			12.6+E[1.60]	1.5		
Shift Register	BSL	bit shift left	10.6+B[0.025]	5.2			10.6+ B[0.025]	5.2		
	BSR	bit shift right	10.9+B[0.025]	5.2			10.9 + B[0.025]	5.2		
	FFL	FIFO load	8.6	3.8			8.7	3.8		
	FFU	FIFO unload	9.5+ E[0.43]	3.7			9.7+ E[0.43]	3.8		
	LFL	LIFO load	8.8	3.7			8.9	3.7		
	LFU	LIFO unload	339	247			10.4	3.8		
Diagnostic	FBC	0 mismatch	372 + B[0.275]	252			14.7 + B[0.055]	2.9		
		1 mismatch	401 + B[0.276]	252			21.4 + B[0.055]	2.9		
		2 mismatches	456 + B[0.272]	252			28.7+ B[0.055]	2.9		

I = execution time of each instruction (operation) used within the FAL or FSC expression

E = number of elements acted on per scan

WI = number of words of memory used by the instruction (operation) within the FAL or FSC expression

Note: FAL or FSC instructions are calculated with short direct addressing

			PLC-5/40, -5/40L, -5/60, -5/60L Series A processors				PLC-5/30 Series A and PLC-5/40, -5/40L, -5/60, -5/60L Series B processors			
	DDT	0 mismatch	373 + B[0.273]	252			14.7 + B[0.055]	2.9		
		1 mismatch	413 + B[0.288]	252			23.4 + B[0.055]	2.9		
		2 mismatches	485 + B[0.375]	252			32.7 + B[0.055]	2.9		
	DTR	data transitional	5.3	5.3			5.3	5.3		
Sequencer	SQI	sequencer input	7.8	1.3			7.8	1.3		
	SQL	sequencer load	7.6	3.5			7.8	3.5		
	SQO	sequencer output	9.6	3.7			9.6	3.7		
Immediate I/O [1]	IIN	immediate input	402	1.1			302	1.1		
	IOT	immediate output	397	1.1			296	1.1		
Zone Control	MCR	master control	0.16	0.16			0.16	0.16		
Program Control	JMP	jump	8.2	1.4			8.9	1.4		
	JSR, RET	jump to subroutine								
	SBR	0 parameters	409	1.1			11.7	1.0		
		1 parameter	433	1.1	441	1.1	15.4	1.0	16.6	1.0
		increase/ parameter	25.2		28.9		3.8		5.0	
	LBL	label	0.16	0.16			0.16	0.16		
	END	end	negligible				negligible			
	TND	temporary end								
	EOT	end of transition								

[1] Timing for immediate I/O instructions is the time for the instruction to queue-up for processing.

B = number of bits acted on per scan;

E = number of words acted on per scan

			PLC-5/40, -5/40L, -5/60, -5/60L Series A processors				PLC-5/30 Series A and PLC-5/40, -5/40L, -5/60, -5/60L Series B processors			
Category	**Code**	**Title**	**Time (µs) integer**		**Time (µs) floating point**		**Time (µs) integer**		**Time (µs) floating point**	
			True	**False**	**True**	**False**	**True**	**False**	**True**	**False**
Program Control	AFI	always false	0.16	0.16			0.16	0.16		
	ONS	one shot	3.0	3.0			3.0	3.0		
	OSR	one shot rising	6.1	6.0			6.2	6.0		
	OSF	one shot falling	6.2	5.8			6.2	5.8		
	FOR/ NXT	for next loop	272 + L[507]	275+ N[5.5]			8.1+ L[15.9]	5.3 + N[0.75]		
	BRK	break	473 + N[5.3]	1.0			11.3 + N[0.75]	0.9		
	UID	user interrupt disable	199	1.0			108	1.0		
	UIE	user interrupt enable	192	1.0			100	1.0		
Process Control	PID	PID loop control								
ASCII [1]	ABL[1]	test buffer for line	482	256			372	150		
	ACB[1]	no. of characters in buffer	483	256			376	150		
	ACI	string to integer	258 + C[20.6]	1.4			140 + C[21.4]	1.4		
	ACN	string concatenate	297 + C[5.5]	1.9			167 + C[5.5]	1.9		
	AEX	string extract	288 + C[1.9]	1.9			159 + C[2.2]	1.9		
	AHL[1]	set or reset lines	663	272			479	150		
	AIC	integer to string	380	1.4			268	1.4		

[1] Timing for ASCII instructions is the time for the instruction to queue-up for processing in channel 0.

C = number of ASCII characters

L = number of FOR/NXT loops

N = number of words in memory between FOR/NXT or BRK/NXT

			PLC-5/40, -5/40L, -5/60, -5/60L Series A processors				PLC-5/30 Series A and PLC-5/40, -5/40L, -5/60, -5/60L Series B processors			
Category	**Code**	**Title**	**Time (µs) integer**		**Time (µs) floating point**		**Time (µs) integer**		**Time (µs) floating point**	
			True	**False**	**True**	**False**	**True**	**False**	**True**	**False**
ASCII	ARD[1]	read characters	491	262			374	144		
	ARL[1]	read line	495	264			377	146		
	ASC	string search	280 + C[2.9]	1.9			151 + C[3.0]	1.9		
	ASR	string compare	285 + C[2.2]	239			169 + C[2.0]	119		
	AWA[1]	write with append	457	266			341	150		
	AWT[1]	write	455	265			339	147		

[1] Timing for ASCII instructions is the time for the instruction to queue-up for processing in channel 0.

C = number of ASCII characters

L = number of FOR/NXT loops

N = number of words in memory between FOR/NXT or BRK/NXT

Timing for PLC-5/10, -5/12, -5/15, -5/25 Processors

Bit and Word Instructions

Table A.C shows timing and memory requirements for bit and word instructions for PLC-5/10, -5/12, -5/15, and -5/25 processors.

Table A.C
Timing and Memory Requirements for Bit and Word Instructions for PLC-5/10, -5/12, -5/15 and -5/25 processors

Category	Code	Title	Execution Time (µs) integer		Execution Time (µs) floating point		Words of Memory**
			True	False	True	False	
Relay	XIC	examine if closed	1.3	0.8			1*
	XIO	examine if open	1.3	0.8			1*
	OTL	output latch	1.6	0.8			1*
	OTU	output unlatch	1.6	0.8			1*
	OTE	output energize	1.6	1.6			1*
Branch		branch end	0.8	0.8			1
		next branch	0.8	0.8			1
		branch start	0.8	0.8			1
Timer and Counter	TON	timer on (0.01 base) (1.0 base)	39 44	27 28			2-3
	TOF	timer off (0.01 base) (1.0 base)	30 30	43 51			2-3
	RTO	retentive timer on (0.01 base) (1.0 base)	 39 44	 24 24			2-3
	CTU	count up	32	34			2-3
	CTD	count down	34	34			2-3
	RES	reset	30	14			2-3

* For every bit address above the first 256 words of memory in the data table, add 0.8 µs to the execution time and 1 word of memory to the requirements.

** Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

Category	Code	Title	Execution Time (µs) integer		Execution Time (µs) floating point		Words of Memory**
			True	False	True	False	
Arithmetic	ADD	add	36	14	92	14	4–7
	SUB	subtract	36	14	92	14	4–7
	MUL	multiply	41	14	98	14	4–7
	DIV	divide	49	14	172	14	4–7
	SQR	square root	82	14	212	14	3–5
	NEG	negate	28	14	36	14	3–5
	CLE	clear	18	14	23	14	2–3
	TOD	convert to BCD	52	14			3–5
	FRD	convert from BCD	44	14			3–5
Logic	AND	and	36	14			4–7
	OR	or	36	14			4–7
	XOR	exclusive or	36	14			4–7
	NOT	not	27	14			3–5
Move	MOV	move	26	14	35	14	3–5
	MVM	masked move	55	14			6–9
Comparison	EQU	equal	32	14	42	14	3–5
	NEQ	not equal	32	14	42	14	3–5
	LES	less than	32	14	42	14	3–5
	LEQ	less than or equal	32	14	42	14	3–5
	GRT	greater than	32	14	42	14	3–5
	GEQ	greater than or equal	32	14	42	14	3–5
	LIM	limit test	42	14	60	14	4–7
	MEQ	mask compare if equal	41	14			4–7

* For every bit address above the first 256 words of memory in the data table, add 0.8 µs to the execution time and 1 word of memory to the requirements.

** Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

Category	Code	Title	Execution Time (µs) integer		Execution Time (µs) floating point		Words of Memory**
			True	False	True	False	
Compute	CPT	add	67	34	124	34	6-9
		subtract	67	34	124	34	6-9
		multiply	73	34	130	34	6-9
		divide	80	34	204	34	6-9
		square root	113	33	244	34	5-7
		negate	59	33	68	34	5-7
		clear	49	30	55	34	4-5
		move	58	33			5-7
		convert to BCD	84	33			5-7
		convert from BCD	75	33			5-7
		AND	68	34			6-9
		OR	68	34			6-9
		XOR	68	34			6-9
		NOT	59	34			5-7
Compare	CMP	equal	63	34	73	34	5-7
		not equal	63	34	73	34	5-7
		less than	63	34	73	34	5-7
		less than or equal	63	34	73	34	5-7
		greater than	63	34	73	34	5-7
		greater than or equal	63	34	73	34	5-7

* For every bit address above the first 256 words of memory in the data table, add 0.8 µs to the execution time and 1 word of memory to the requirements.

** Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

File Instructions

The instruction timing for file instructions depends on the data type, number of files acted on per scan, number of elements acted on per scan, and whether the instruction converts data between integer and floating point formats.

- for integer to floating point conversion, add:

  8 µs for each element address
  10 µs for each file address (# prefix)

- for floating point to integer conversion add:

  33 µs for each element address
  44 µs for each file address (# prefix)

Table A.D
Timing and Memory Requirements for File Instructions for PLC-5/10, -5/12, -5/15 and -5/25

Category	Code	Title	Time (µs) integer True	Time (µs) floating point True	Time (µs) integer or floating point False	Words of Memory*
File Arithmetic and Logic	FAL	add	98 + W[36.7 + N]	98 + W[95.1 + N]	54	7–12
		subtract	98 + W[36.7 + N]	98 + W[95.1 + N]	54	7–12
		multiply	98 + W[42.5 + N]	98 + W[101.2 + N]	54	7–12
		divide	98 + W[51.1 + N]	98 + W[180.3 + N]	54	7–12
		square root	98 + W[84.7 + N]	98 + W[220.5 + N]	54	6–10
		negate	98 + W[29.2 + N]	98 + W[37.2 + N]	54	6–10
		clear	98 + W[18.4 + N]	98 + W[24.0 + N]	54	5–8
		move	98 + W[27.3 + N]	98 + W[36.2 + N]	54	6–10
		convert to BCD	98 + W[54.3 + N]		54	6–10
		convert from BCD	98 + W[45.4 + N]		54	6–10
		AND	98 + W[37.2 + N]		54	7–12
		OR	98 + W[37.2 + N]		54	7–12
		XOR	98 + W[37.2 + N]		54	7–12
		NOT	98 + W[28.2 + N]		54	6–10
File Search and Compare	FSC	all comparisons	93 + W[32.7 +N]	93 + W[43.3 +N]	54	6–10

W = number of elements acted on per scan

N = 2 x (number of integer file addresses) + 8 x (number of floating-point file addresses) + 6 x (number of timer, counter, or control file addresses) + (number of conversions between integer and floating point formats)

* Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

Category	Code	Title	Time (µs) integer True	Time (µs) floating point True	Time (µs) integer or floating point False	Words of Memory*
File	COP	copy counter, timer, and control	88 + 2.7W 98 + 5.8W	104 + 3.8W	20	4–7
	FLL	fill counter, timer, and control	81 + 2/.1 W 97 + 4.4W	100 + 3.1W	15	4–7
Shift Register	BSL	bit shift left	74 + 3.4W		57	4–7
	BSR	bit shift right	78 + 3.0W		57	4–7
	FFL	FIFO load	54		44	4–7
	FFU	FIFO unload	68 + 3.2W		46	4–7
Diagnostic	FBC	file bit compare				6–11
		0 mismatch	75 + 6W		31	
		1 mismatch	130 + 6W		31	
		2 mismatches	151 + 6W		31	
	DDT	diagnostic detect				6–11
		0 mismatch	71 + 6W		31	
		1 mismatch	150 + 6W		31	
		2 mismatches	161 + 6W			
Zone Control	MCR	master control	12		18	1
Immediate I/O	IIN	immediate input				2–3
		local	196		16	
		remote	204		16	
	IOT	immediate output				2–3
		local	202		16	
		remote	166		16	

W = number of elements acted on per scan

N = 2 x (number of integer file addresses) + 8 x (number of floating-point file addresses) + 6 x (number of timer, counter, or control file addresses) + (number of conversions between integer and floating point formats)

* Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

Category	Code	Title	Time (µs) integer True	Time (µs) floating point True	Time (µs) integer or floating point False	Words of Memory*
Sequencer	SQI	sequencer input	57		14	5–9
	SQL	sequencer load	55		42	4–7
	SQO	sequencer output	77		42	5–9
Jump and Subroutine	JMP	jump	45		15	2–3
	JSR SBR	jump to subroutine				
		0 parameters	56		15	2–3
		1 parameter	91		15	3–5
		add per parameter	21			
	RET	return from sub.				
		0 parameters	48		13	1
		1 parameter	70		13	2–3
		add per parameter	21			
	LBL	label	12		12	3
Miscellaneous	END	end	negligible		negligible	1
	TND	temporary end	negligible		15	1
	AFI	always false	15		13	1
	ONS	one shot	28		30	2–3
	DTR	data transitional	41		41	4–7
	BTD	bit distributor	77		14	6–11
	PID	PID loop control	608		34	5–9
	BTR BTW	block transfer read block transfer write	See Block Transfer BTR, BTW chapter			
	MSG	message	See Message MSG chapter			

W = number of elements acted on per scan

N = 2 x (number of integer file addresses) + 8 x (number of floating-point file addresses) + 6 x (number of timer, counter, or control file addresses) + (number of conversions between integer and floating point formats)

* Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

Program Constants

Use program constants in compare, compute, and file instructions to improve instruction execution times. Integer constants and floating-point constants execute in less than 1 µs.

Direct and Indirect Elements PLC-5/30, -5/40, -5/40L, -5/60 and -5/60L Processors

Additional execution time for directly and indirectly addressed elements depends on location in memory, reference to the beginning of all data files (output file, word 0), whether data is stored at the source or destination address, and whether the instruction converts data. Table A.E lists times to add to instruction execution times.

Table A.E
Additional Execution Time

Addressing Mode	Data Type	Modifier in µsec (add for each operand)
Direct	Integer	0
	Float	0
Index	Integer	1.1
	Float	1.8
	CTC	2.4
Immediate	Integer	0.24
	Float	1.0
Indirect		6.6 + W[0.09]
Float-to-integer		5.6
Integer-to-float		8.4

Direct and Indirect Elements PLC-5/10, -5/12, -5/15 and -5/25 Processors

Additional execution time for directly addresses elements depends on location in memory, reference to the beginning of all data files (output file, word 0), whether data is stored at the source or destination address, and whether the instruction converts data. Table A.F lists times to add to instruction execution times.

Table A.F
Additional Execution Time Based on Source and Destination Addresses
PLC-5/10, -5/12, -5/15, -5/25 Processors

Data Type	Source (integer to floating point)			Destination (floating point to integer)		
	0–2K	2–4K	4K+	0–2K	2–4K	4K+
integer	0	1	2	0	1	2
floating point	0	3	4	0	3	4
data conversion	8	9	10	33	34	35

When file addresses (# prefix) in the expression or destination address contain indirect addresses for file numbers, add:

- 45 µs when the indirect address is integer type
- 48 µs when the indirect address is floating point type
- 48 µs when the indirect address is timer, counter, or control type

When file addresses in the expression or destination contain indirect addresses for element numbers, add:

- 45 µs when the indirect address is integer type
- 46 µs when the indirect address is floating point type
- 46 µs when the indirect address is timer, counter, or control type

If the file address contains two indirect addresses, add only one value (the largest). For example, for #F[N7:20][N7:30], add 48 µs (indirect floating point file address).

Multiply the additional time by the number of elements in the file for any type of file or file address. For example:

Expression: #N[N7:100]/10 * F8:20
add 10 for converting to floating point
add 45 for indirect address

Destination: #N7:30
add 44 for convering to integer

FAL multiply: 98 + W[42.5 + N + indirect addressing]
N = 2(2) +8 (1) + 6(0) + 10 + 44 = 66
W = 16

Execution time in ALL mode:
98 + 16[42.5 + 66 +45]
2554 µs

Appendix B

SFC Reference

Chapter Objectives

Use this chapter to make sure your SFC meets your processor's requirements and to make sure your SFC runs the way you expect. This chapter discusses:

- SFC status information in the Processor Status file
- Memory allocation
- Dynamic constraints
- Scanning sequences
- Run times

SFC Status Information in the Processor Status File

Table B.A lists the words and bits in the processor status file (S) that contain SFC information.

Table B.A
SFC Status Words

Word:	Title:	Description:
S:1/15	First pass	Set: Processor began first program scan of the next active step in the SFC Reset: Processor completed scanning the currently active step
S:8	Current program scan time	The time for the processor to scan through all active steps one time If you are using multiple main programs on a PLC-5/30, -5/40 or -5/60, this time is the total of all main programs.
S:9	Maximum program scan time	The maximum time for the processor to scan through all active steps one time (word 8) If you are using multiple main programs on a PLC-5/30, -5/40 or -5/60, this time is the total of all main programs.
S:11/3	SFC fault	Set: Processor detected an SFC fault and stored a fault code in word 12 Reset: No SFC fault
S:11/5	Start up fault	Set: Processor detected a start-up protection fault (see word 26 bit 1) Reset: No fault, start up allowed

Word:	Title:	Description:
S:12	Fault codes	74 Fault in SFC file 75 SFC has more than 24 active steps 77 Missing file or file of wrong type for step, action or transition 78 SFC execution cannot continue after interruption 79 Cannot run SFC because PLC-5 is incompatible
S:13	Faulted File Number	Contains the file number if an SFC fault occurred
S:14	Faulted Rung Number	Contains the faulted rung number
S:26/0 *	Restart/continue	Set: Processor restarts SFC at the active steps where it left off due to power loss or processor mode change Reset: Processor restarts SFC at first step
S:26/1	Start-up protection after power loss	Set: Protection enabled; processor goes to fault routine at power up and processor sets word 11, bit 5 Reset: Protection disabled; processor powers up in run mode
S:28 *	Program watchdog setpoint	Maximum time (milliseconds) for scanning a single pass through all active steps If you are using multiple main programs on a PLC-5/30, -5/40 or -5/60, this time is the total of all main programs.
S:79 – S:127	MCP inhibit, file number and scan time	Information on the individual multiple main programs. PLC-5/30, -5/40 or -5/60 only.

* You enter values for these words

Memory Allocation

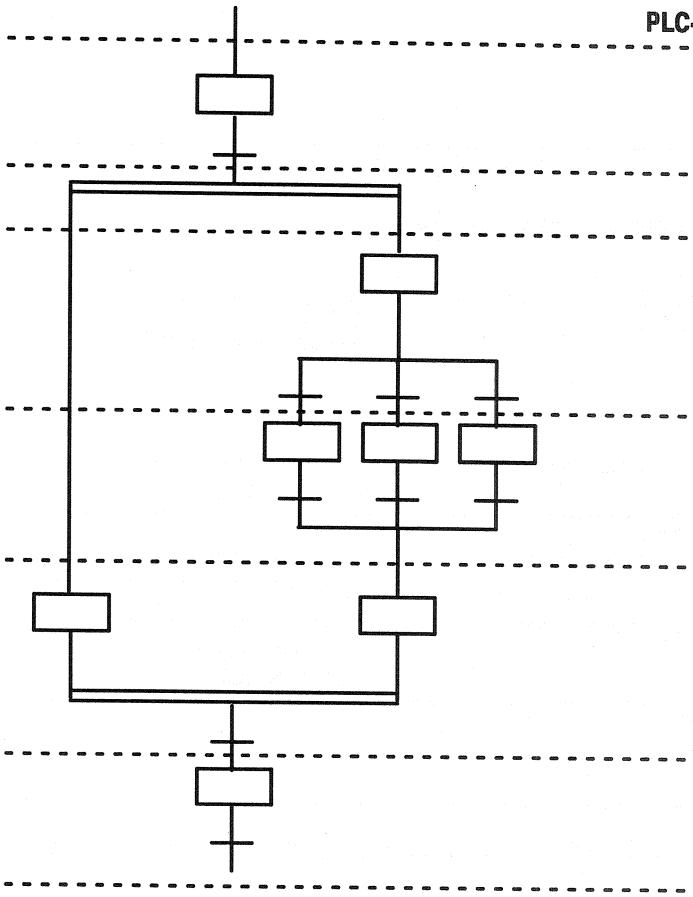
The memory requirements for your SFC depend on the structures you use. Table B.B shows estimated word usage for SFC structures:

Table B.B
SFC Memory Usage

This Structure:	Uses this Amount of Memory:	
	PLC-5/10, -5/12, -5/15, -5/25	PLC-5/30, -5/40, -5/40L, -5/60, -5/60L
start and end of program	2 words	19 words
each step /transition pair	8 words	16 + 16*a* words *a* = number of actions in step 6 words each action
each selection branch	5*n* + 5 words *n* = number of branches	11 + 6*a* + 7*n* *a* = number of actions in step *n* = number of paths
each simultaneous branch, diverging	*n* + 1 word *n* = number of branches	3*n* + 1 *n* = number of paths
each simultaneous branch, converging	n^2 + 6*n* + 3 words *n* = number of branches	5 + 11*n* + 6*a* *a* = number of actions in all converging steps for that simultaneous branch *n* = number of paths
each label or GOTO statement	1 word	1 word
each chart compression	3 words	3 words

Figure B.1 shows a sample SFC and the estimated memory requirements for the SFC.

Figure B.1
Sample SFC and Memory Requirements



	PLC-5/10, -5/12, -5/15, -5/25	PLC-5/30, -5/40, -5/40L, -5/60, -5/60L
		one action/step a = 1 16 + 6a = 22 words
	step/transition pair 8 words	
	simultaneous diverge n = 2 n + 1 = 3 words	simultaneous diverge n = 2 3n + 1 = 7 words
	selection branch n = 3 5n + 5 = 20 words	selection branch n = 3 11 + 6a + 7n = 38 words
	3 step/transition pairs 3 x 8 = 24 words	3 step/transition pairs 3 (16 + 6a) = 66 words
	simultaneous converge n = 2 n^2 + 6n + 3 = 19 words	simultaneous converge n = 2 5 + 11n + 6a = 39 words
	step/transition 8 words	one action/step 16 + 6a = 22 words
	82 words (sub total) + 2 words (start and end of program)	194 words (sub total) + 17 words (start and end of program) (8 actions * 6 words - assumes 1 unique action per step)
	84 words total for SFC	259 words total for SFC

Dynamic Constraints

The PLC-5/30, -5/40 and -5/60 processors have no dynamic constraints.

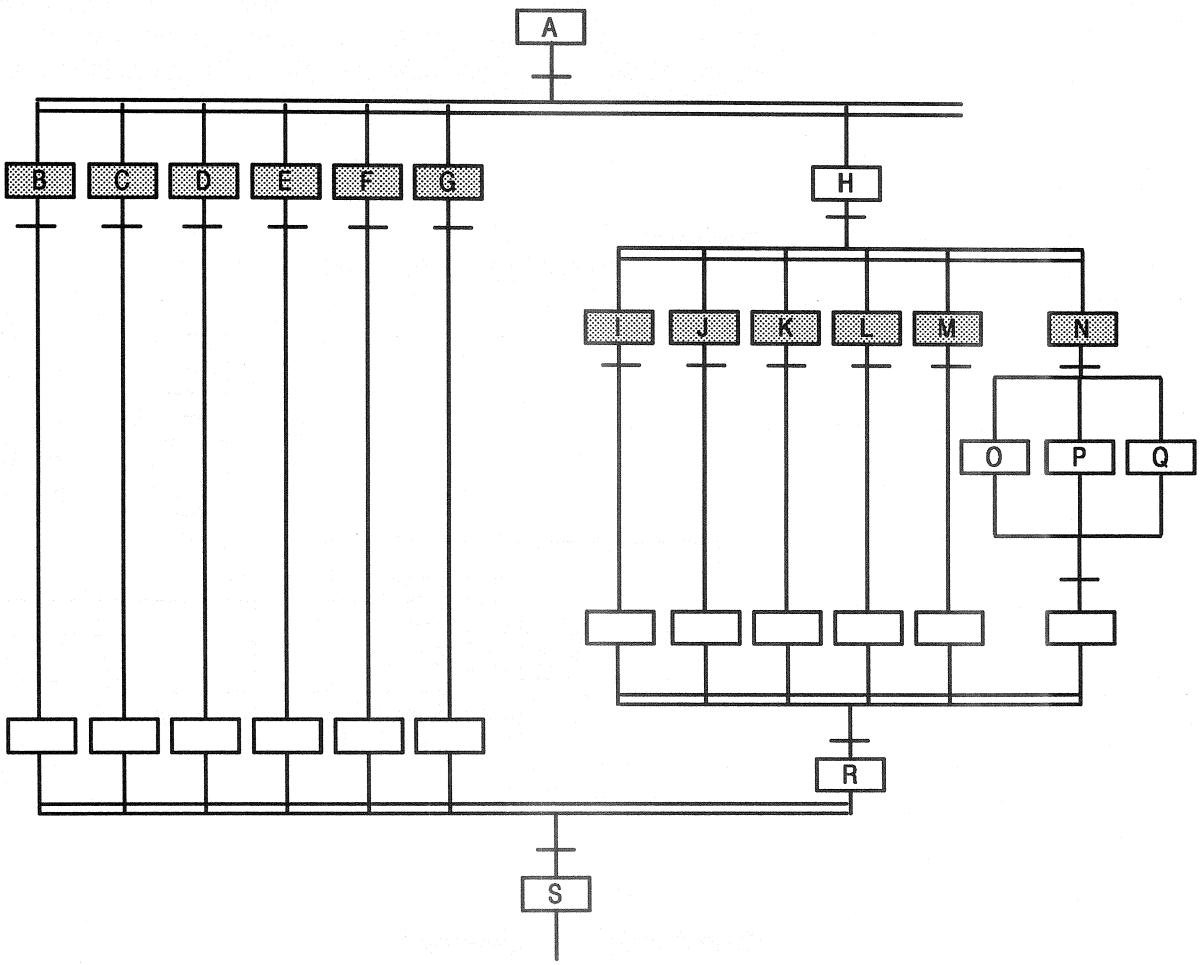
If you are using a PLC-5/10, -5/12, -5/15, -5/25 processor and your SFC has more than 12 parallel paths, you need to determine the number of parallel paths that could be active at one time. The dynamic limit is 24 parallel paths active at the same time for a PLC-5/10, -5/12, -5/15, or -5/25 processor.

When a transition goes true, momentarily both the previously active step(s) (now waiting for postscan) and the newly active step(s) are on the execution queue together. You can have up to 23 parallel active steps as long as you can guarantee that no more than one transition goes true at one time.

Determine the number of active steps by counting the steps on each side of the transitions that control the widest area of the SFC. For example, 12 transitions that are true at the same time account for at least 24 simultaneous active steps. If any new simultaneous divergences follow one of these transitions, the maximum of 24 active paths is exceeded.

If the function chart in Figure B.2 is at the point where all 12 shaded steps are active and all of the transitions following those steps become true at the same time, the system attempts to have 26 active steps (12 for postscan, 14 for first scan) and the processor will fault.

Figure B.2
Dynamic Limit of Active Steps Could Be Exceeded
(PLC-5/10, -5/12, -5/15, -5/25 Processors)



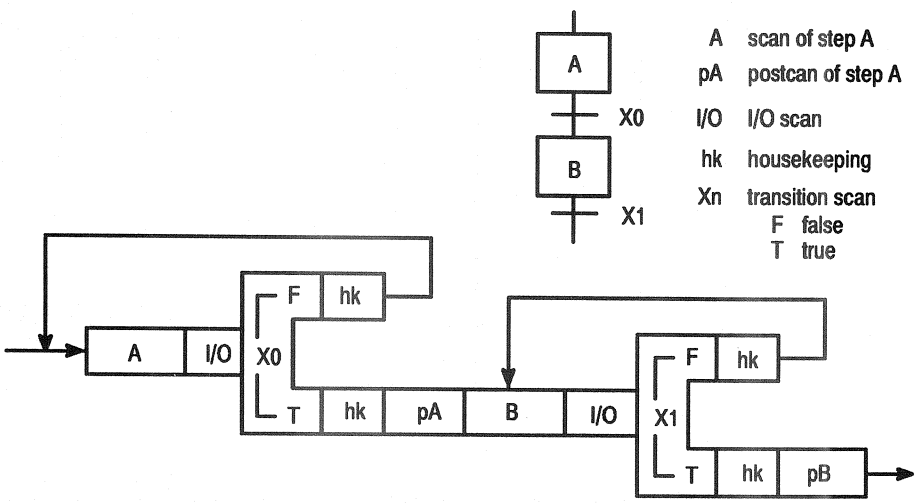
Scanning Sequences

The processor scans the SFC from top to bottom, left to right. When the scan encounters active parallel steps, the processor runs the ladder logic in the left-most step first, then moves to the ladder logic in the next parallel step, until all active steps are run. The processor recognizes parallel steps by their position with respect to their common divergence, not necessarily by their position on the screen.

Step and Transition Scanning

In general, the processor scans an active step, then scans the I/O, and then continues this cycle until the transition logic is true. When the transition is true, the processor scans the current step one more time (postscan). During postscan, the processor forces all rungs in the step false and resets rung logic. The processor does not update I/O between a postscan and the scan of the next active step. Figure B.3 shows the scan sequence for a step, transition and postscan. If you are using PLC-5/30, -5/40 or -5/60 processors, you can configure the scan and postscan operations. For more information, refer to the "Editing SFCs" chapter in the Entering and Editing SFCs section of the Programming module.

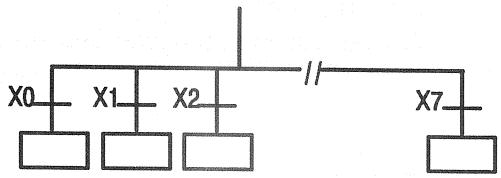
Figure B.3
Scan Sequence for a Step, Transition, and Postscan



Selected Branch Scanning

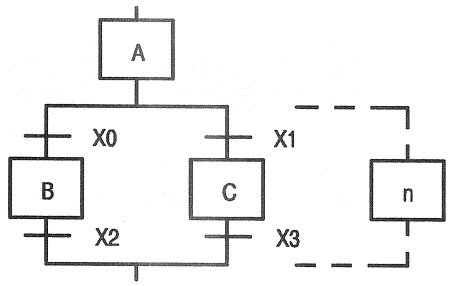
The processor selects one path of multiple parallel paths in a selected branch (Figure B.4). The processor tests transitions X0 through Xn, from left to right, until one of the transitions become true. The path with the first true transition is the active path.

Figure B.4
Selected Branch - Divergence



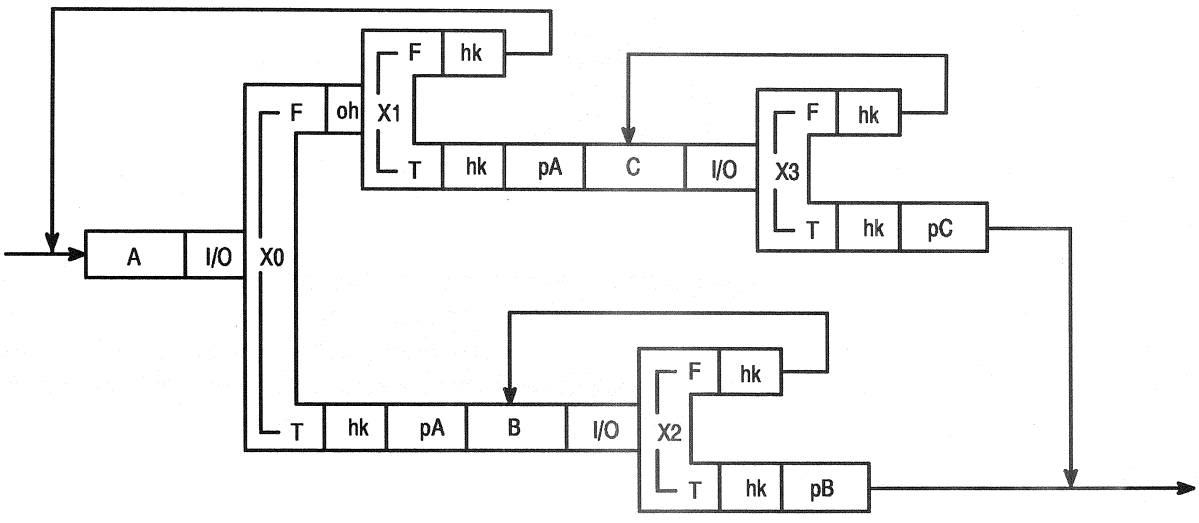
Because only one path is active, the scan sequence for the convergence is the same as for a step and transition. Figure B.5 shows the scan sequence for the divergence and convergence of a selected branch.

Figure B.5
Scan Sequence for a Selected Branch - Divergence and Convergence



PLC-5/10, -5/12, -5/15, -5/25: maximum of 7 selections
PLC-5/30, -5/40, -5/40L, -5/60, -5/60L: maximum of 16 selections

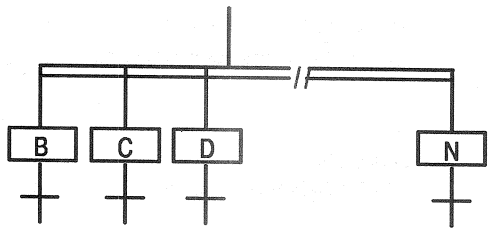
A	scan of step A
pA	postcan of step A
I/O	I/O scan
hk	housekeeping
oh	overhead
Xn	transition scan
F	false
T	true
n	transition number



Simultaneous Branch Scanning

The processor scans all parallel paths in a simultaneous branch (Figure B.6). On the first scan, the processor scans step B, then step C, until the processor scans all the steps on the divergence.

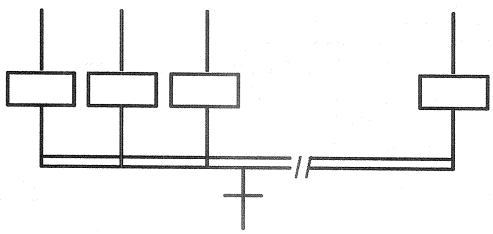
Figure B.6
Simultaneous Branch - Divergence



On subsequent scans, the processor scans in the order of step, I/O, and transition for each path, starting from the left.

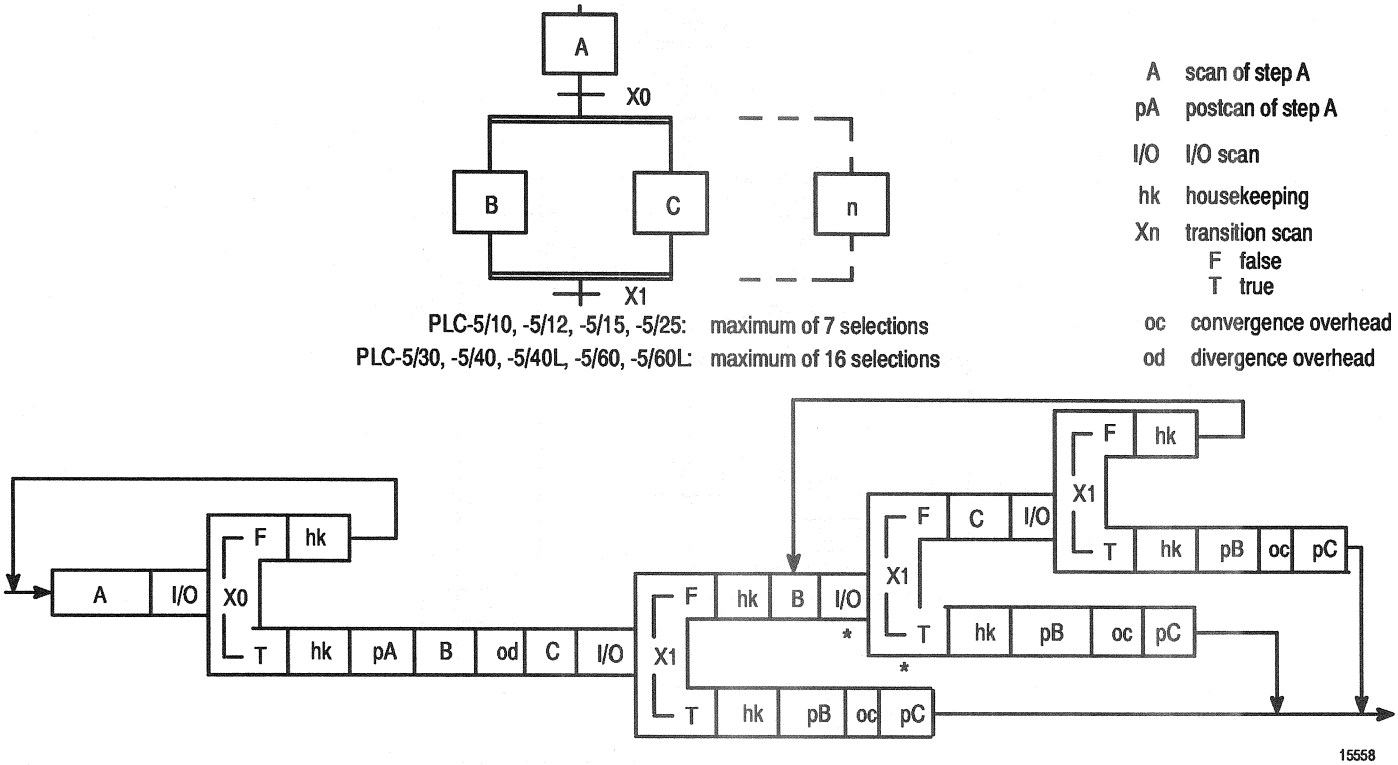
The vertical progression from step to step is independent of the active steps on the other parallel paths (Figure B.7).

Figure B.7
Simultaneous Branch - Convergence



The common transition cannot go true until the processor scans all the steps in the simultaneous branch at least once. Once the transition goes true, the processor does not scan the remaining paths in the branch; the processor postscans each step in the branch. Figure B.8 shows the scan sequence for the divergence and convergence of a selected branch.

Figure B.8
Scan Sequence for a Simultaneous Branch - Divergence and Convergence



* In a PLC-5/30, -5/40, -5/40L, -5/60, -5/60L, these states do not occur if scan confirmation is set to 'ADVANCED' mode

SFC Example and Scan Sequence

Figure B.9 shows an example SFC. Figure B.10 shows the scan sequence for the example SFC. Use this example SFC and scan sequence as a guide. These figures may not apply to your system.

Figure B.9
Example SFC for Scan Sequence Example

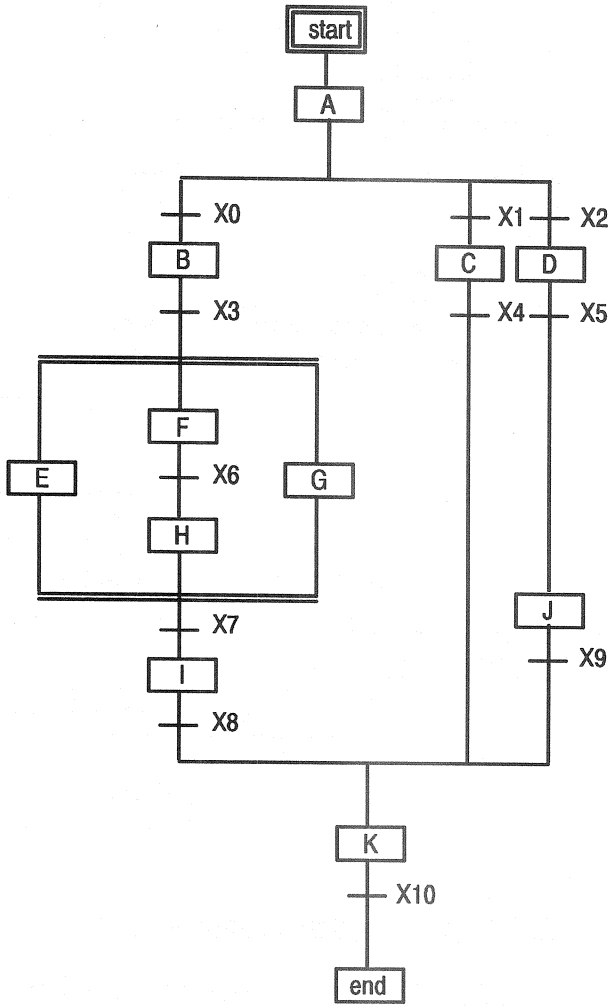
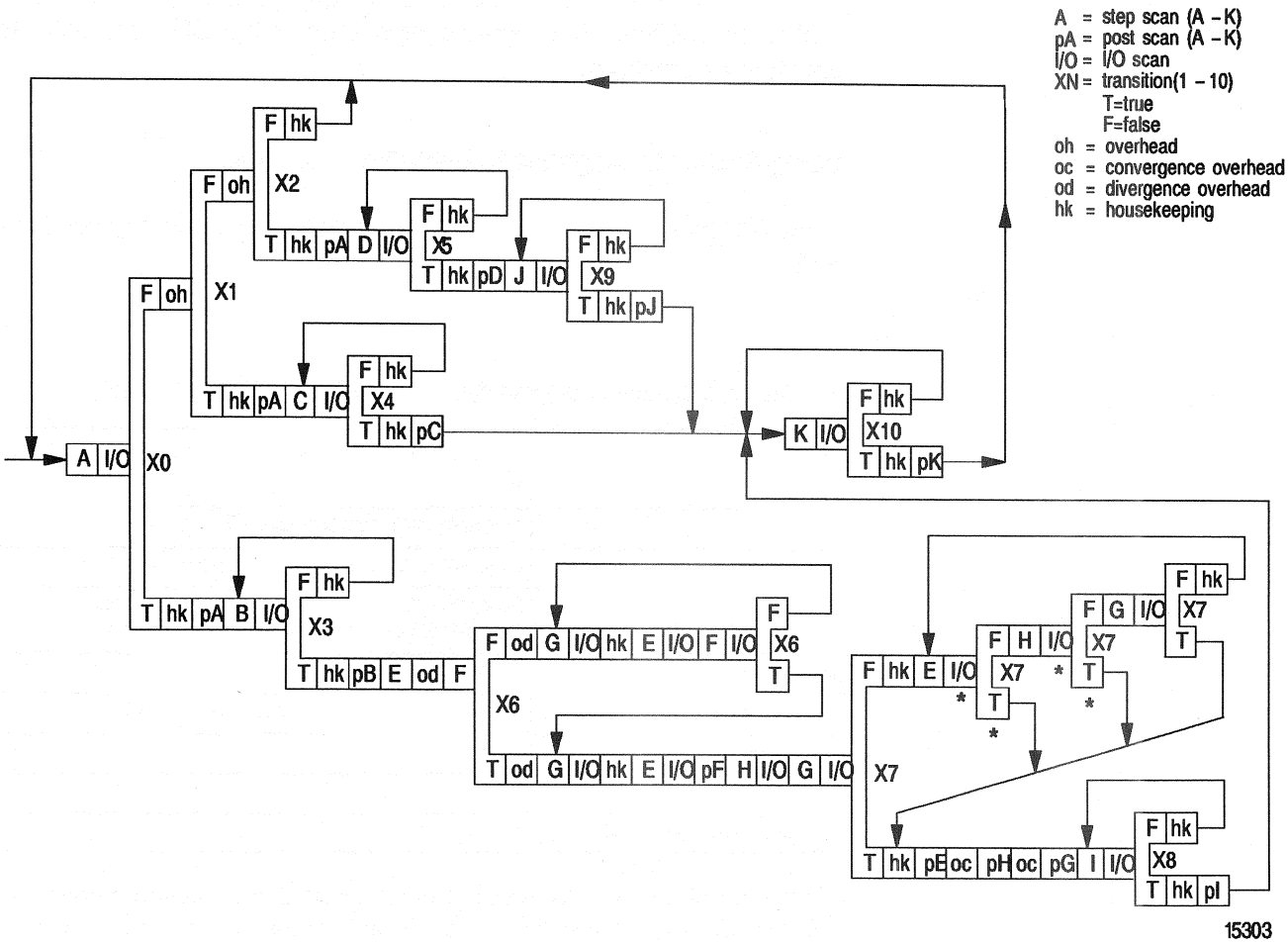


Figure B.10
Scan Sequence Example for the Example SFC



* In a PLC-5/30, -5/40, -5/60, these states do not occur if scan confirmation is set to 'ADVANCED' mode

Run Times PLC-5/10, -5/12, -5/15, -5/25

To determine the run time of your processor memory file on a PLC-5/10, -5/12, -5/15, -5/25, you add the run time for ladder logic and the run time for the SFC. For information about run times for ladder logic, see the "Instruction Set Reference" module. To determine the run time for an SFC, use either sequence diagrams or equations.

Using Sequence Diagrams to Determine Run Time

Table B.C lists the run times to add based on the sequence diagram for your SFC.

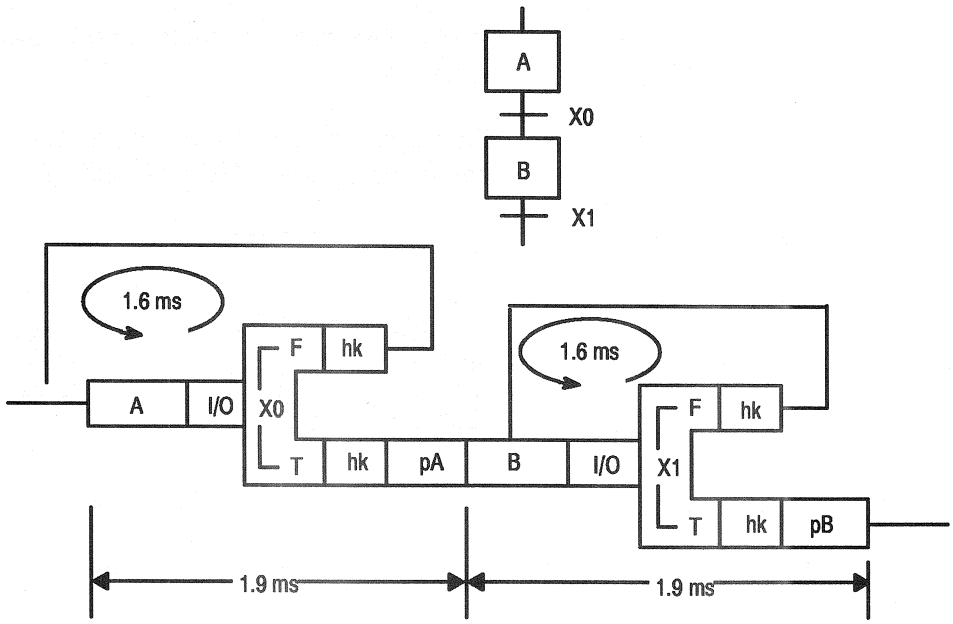
Table B.C
Run Times for Sequence Diagram Sections - PLC-5/10, -5/12, -5/15, -5/25

This Event:	Takes this Amount of Time (in milliseconds):
A	time to execute logic of step A + 0.1 ms
pA	time to scan logic of step A with rungs false + 0.1 ms
XN	transition N false (F): time to scan logic + 0.1. ms transition N true (T): time to scan logic + .25 ms
I/O (I/O scan)	0.6 ms
hk (housekeeping)	0.7 ms (increases with increasing DH+ traffic)
oh (overhead)	0.02 ms
od (divergence overhead)	0.3 ms
oc (convergence overhead)	0.2 ms

To determine the worst-case run time, assume that a transition goes true just after an I/O scan or just after a transition is scanned. This assumption requires an extra scan sequence before the transition goes true.

The scan time of a step and transition is proportional to the number of rungs for the step and transition. Figure B.11 shows the minimum scan time for a step that contains a single OTE and an END statement and a transition that contains a single XIC and an EOT statement.

Figure B.11
Minimum Scan Time for a Step and Transition Pair



Using Equations to Determine Run Time

The equations you use depend on whether the scan is steady state (simple step and transition) or divergent and convergent.

Steady-state Scan Time is when all transitions following active steps are false. Use this equation (Table B.D):

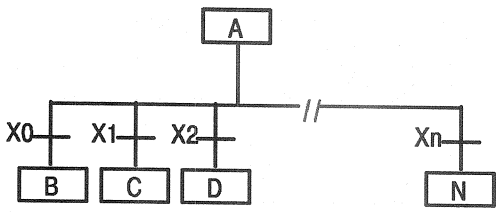
$$T_{milliseconds} = 0.8a + 0.7 + T_{scan}$$

Table B.D
Variables for Steady-State Scan Time

Where:	Is:
$T_{milliseconds}$	steady-state scan time in milliseconds
a	number of active steps
T_{scan}	total time to scan logic in all active steps and associated false transitions

Divergent Scan Time starts when the processor tests a transition and ends when the processor scans the next step's I/O. Divergent scan time includes transition scan time, postscan time of the previous step, scan time of the new step, overhead, and scan time of each parallel active step outside of the divergence.

For a selected-path divergence, the best case is when the transition goes true just before the I/O scan. Use this equation (Table B.E):

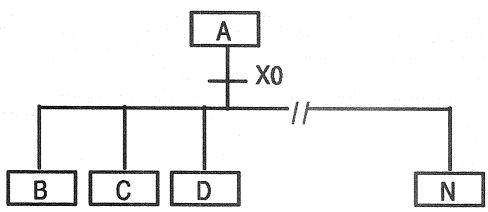


$$T_{milliseconds} = T_X + pA + T_S + 0.02(n-1) + 1.55 + 0.8a + T_O$$

Table B.E
Variables for Selected-Path Divergent Scan Time

Where:	Is:
$T_{milliseconds}$	transition scan time in milliseconds from step A to the first step in selected path N
T_X	sum of scan times of logic of transitions X0, X1, ..., Xn in the divergence, up to and including the selected transition
pA	postscan time for the step (step A) preceding the divergence
T_S	scan time for logic in the new step (step N)
n	path number selected (1–7, from left to right)
a	number of active steps outside the divergence
T_O	sum of scan times of logic in all other active steps and transitions parallel to the divergence, but outside of the divergence

For a simultaneous divergence, the best case is when the transition goes true just before the I/O scan. Use this equation (Table B.F):



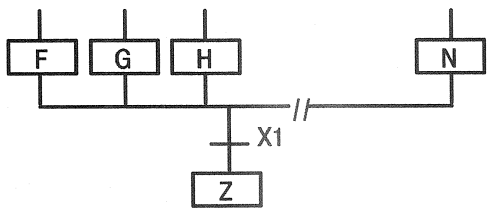
$$T_{milliseconds} = T_{X0} + pA + T_S + 0.3(n-1) + 1.97 + 0.8a + T_O$$

Table B.F
Variables for Simultaneous-Path Divergent Scan Time

Where:	Is:
$T_{milliseconds}$	transition time in milliseconds from when transition X0 goes true until the processor finishes scanning the last simultaneous step (step N) in the divergence
T_{X0}	scan time of logic in transition X0
pA	time to do a post-scan of step A
T_S	sum of scan times of logic in new steps (step B, step C, ..., step N)
n	number of simultaneous active steps in the divergence
a	number of parallel active steps outside the divergence
T_0	sum of scan times of logic in all other active steps and transitions parallel to the divergence, but outside of the divergence

For the worst case, assume that a transition goes true just after the I/O scan or just after a transition is scanned. This assumption requires an extra scan sequence before the transition goes true.

Convergent Scan Time is when a simultaneous branch ends. The best case is when the transition goes true just before the I/O scan. Use this equation (Table B.G):



$$T_{milliseconds} = T_{X1} + T_p + T_Z + 0.2(n-1) + 1.5 + 0.8a + T_0$$

Table B.G
Variables for Simultaneous-Path Convergent Scan Time

Where:	Is:
$T_{milliseconds}$	transition time in milliseconds from when transition X1 goes true until the processor finishes scanning step Z
T_{X1}	scan time of logic in transition X1
T_p	sum of postscan times of steps F, G, ..., N
T_Z	scan time of logic in step Z
n	number of simultaneous active steps in the convergence
a	number of parallel active steps outside of the convergence
T_0	sum of scan times of logic in all other active steps and transitions parallel to the convergence, but outside of the convergence

For the worst case, assume that a transition goes true just after the I/O scan or just after a transition is scanned. This assumption requires an extra scan sequence before the transition goes true.

Appendix C

Valid Data Types for Instruction Operands

Chapter Objectives

This chapter lists all of the available instructions and their operands and the data types/values that are valid for each operand.

The following table explains each valid data type/value:

This Data Type/Value:	Accepts:
immediate (program constant)	any value between -32768 and 32767
integer	any integer data type: integer, timer, counter, status, bit, input, output, ASCII, BCD, control (e.g., N7:0, C4:0, etc.)
float	any floating point data type (valid range is $\pm$ 1.1754944e^{-38} to $\pm$3.4028237e^{+38}). Note that the 1784-T50 may round or truncate the number based on the number of signigicant digits and the size of the floating point number.
block	any block transfer data type (e.g., BT14:0) or integer data type (e.g.N7:0)
message	any message data type (e.g., MG15:0) or integer data type (e.g.N7:0)
PID	any PID data type (e.g., PD16:0) or integer data type (e.g.N7:0)
string	any string data type (e.g., ST12:0)
SFC status	any SFC status data type (e.g., SC17:0)

Instruction Operands and Valid Data Types

Table C.A shows the programming instructions you can use and the operands for those instructions. You can also use this table to format instructions in ASCII for importing. For more information on importing, see the "Working with Files" section of the "Programming" module.

Instructions marked with an asterisk (*) are only supported by PLC-5/30, -5/40. -5/40L, -5/60 and -5/60L processors.

Table C.A
Programming Instructions and Operands

Instruction	Description	Operand	Valid Value
ABL *	ASCII Test Buffer for Line	channel	immediate, integer 0–4
		control	control
ACB *	ASCII Number of Characters in Buffer	channel	immediate, integer
		control	control
ACI *	ASCII String to Integer	source	string
		destination	integer
ACN *	ASCII String Concatenate	source A	string
		source B	string
		destination	string
ACS *	Arc Cosine	source	immediate, float (in radians)
		destination	immediate, float (in radians)
ACT *	SFC action (only for ASCII import/export)	action number	immediate
		file number	0 - 999
		destination	string
ADD	ADD	source A	immediate, integer, float
		source B	immediate, integer, float
		destination	integer, float
AEX *	String Extract	source	string
		index	immediate, integer
		number	immediate, integer
		destination	string
AFI	Always False	none	
AHL *	ASCII Set/Reset Handshake Lines	channel	immediate, integer
		handshake AND mask	immediate, integer
		handshake OR mask	immediate, integer
		control	control
AIC *	ASCII Integer to String	source	immediate, integer
		destination	string

Instruction	Description	Operand	Valid Value
AND	Logical AND	source A	immediate, integer
		source B	immediate, integer
		destination	integer
ARD *	ASCII Read Characters	channel	immediate, integer
		destination	string
		control	control
		string length	0 - 82
ARL *	ASCII Read Line	channel	immediate, integer
		destination	string
		control	control
		string length	0 - 82
ASC *	ASCII String Search	source	string
		index	immediate, integer
		search	string
		result	integer
ASN *	Arc Sine	source	immediate, float (in radians)
		destination	immediate, float (in radians)
ASR *	ASCII String Compare	source A	string
		source B	string
ATN *	Arc Tangent	source	immediate, float (in radians)
		destination	immediate, float (in radians)
AVE *	Average File	file	integer, float
		destination	integer, float
		control	control
		length	1 - 1000
		position	0 - 999
AWA *	ASCII Write with Append	channel	immediate, integer
		source	string
		control	control
		string length	0 - 82

Instruction	Description	Operand	Valid Value
AWT *	ASCII Write	channel	immediate, integer
		source	string
		control	control
		length	0 - 82
BRK	Break	none	
BSL	Bit Shift Left	file	binary
		control	control
		bit address	bit
		length	1 - 16000 (length in bits)
BSR	Bit Shift Right	file	binary
		control	control
		bit address	bit
		length	1 - 16000 (length in bits)
BTD	Bit Distribute	source	immediate, integer
		source bit	immediate, integer (0 - 15)
		destination	integer
		destination bit	immediate, integer (0 - 15)
		length	immediate, integer (1 - 16)
BTR	Block Transfer Read	rack	00-277 octal
		group	0-7
		module	0-1
		control block	block, integer
		data file	integer
		length	0, 1-64
		continuous	YES, NO
BTW	Block Transfer Write	rack	00-277 octal
		group	0-7
		module	0-1
		control block	block, integer
		data file	integer
		length	0, 1-64

Instruction	Description	Operand	Valid Value
		continuous	YES, NO
CIR	Custom Input Routine (for use with CAR applications only)	program file number	immediate (2 - 999)
		input parameter list	immediate, integer, float
		return parameter list	integer, float
CLR	Clear	destination	integer, float
CMP	Compare	expression, relative expression, expression	expression using values or addresses with evaluators (for a list, see chapter 3 in this module)
		EXE mnemonic (end of expression) only for ASCII import	EXE
COP	File Copy	source	array
		destination	array
		length	immediate (1 - 1000)
COR	Custom Output Routine (for use with CAR applications only)	program file number	immediate (2 - 999)
		input parameter list	immediate, integer, float
		return parameter list	integer, float
COS *	Cosine	source	immediate, float (in radians)
		destination	immediate, float (in radians)
CPT	Compute	math expression	expression using values or immediate integer float addresses with evaluators (for a list, see chapter 4 in this module)
		EXE mnemonic only for ASCII import	EXE
		mode word	immediate, integer (0, INC, 1-1000, 2048, ALL)
		relative expression	addresses with evaluators (for a list, see chapter 4 in this module)
		destination	integer float
CTD	Count Down	counter	counter
		PRE	-32768 - +32767
		ACC	-32768 - +32767

Instruction	Description	Operand	Valid Value
CTU	Count Up	counter	counter
		PRE	-32768 - +32767
		ACC	-32768 - +32767
DDT	Diagnostic Detect	source array	binary
		reference array	binary
		result array	integer
		compare control	control
		length	1 - 16000 (length in bits)
		position	0 - 15999
		result control	control
		length	1 - 1000
		position	0 - 999
DEG *	Degree (convert radians to degrees)	source	immediate, float (in radians)
		destination	immediate, float (in degrees)
DIV	Divide	source A	immediate, integer, float
		source B	immediate, integer, float
		destination	integer, float
DTR	Data Transitional	source	immediate, integer
		mask	immediate, integer
		reference	integer
EOC	end of SFC compression (see SOC)	only for ASCII import/export	
EOR	end of rung	only for ASCII import/export	
EOT	end of transition	none	
ESE	end SFC selection branch (see SEL)	only for ASCII import/export	
EQU	Equal	source A	immediate, integer, float
		source B	immediate, integer, float
EOP	end of SFC program	only for ASCII import/export	
ERI	error on an input instruction	only in ASCII export files	
ERO	error on an output instruction	only in ASCII export files	

Instruction	Description	Operand	Valid Value
ESI	end SFC simultaneous branch (see SIM)	only for ASCII import/export	
FAL	File Arithmetic/Logical	control	control
		length	1 - 1000
		position	0 - 999
		mode	immediate, integer (0, INC, 1-1000, 2048, ALL)
		destination	integer
		math expression	indexed math instruction
FBC	File Bit Compare	source array	binary
		reference array	binary
		result array	integer
		compare control	control
		length	1 - 16000 (length in bits)
		position	0 - 15999
		result control	control
		length	1 - 1000
		position	0 - 999
FFL	FIFO Load	source operand	immediate, indexed, integer
		FIFO array	integer
		FIFO control	control
		length	1 - 1000
		position	0 - 999
FFU	FIFO Unload	FIFO array	integer
		destination	indexed, integer
		FIFO control	control
		length	1 - 1000
		position	0 - 999
FLL	Fill File	source operand	immediate, integer, float
		destination array	array
		length	immediate (1 - 1000)

Instruction	Description	Operand	Valid Value
FOR	For Loop	LBL number	integer
		index	integer
		initial value	immediate, integer
		terminal value	immediate, integer
		step size	immediate, integer
FRD	From BCD	source	immediate, integer
		destination	integer
FSC	File Search and Compare	control	control
		length	1 - 1000
		position	0 - 999
		mode	immediate, integer (0, INC, 1-1000, 2048, ALL)
		math expression	indexed math instruction
GEQ	Greater Than or Equal To	source A	immediate, integer, float
		source B	immediate, integer, float
GRT	Greater Than	source A	immediate, integer, float
		source B	immediate, integer, float
IIN	Immediate Input	I/O word	immediate, integer PLC-5/10, 12, 15, 25, 30: 000-077 PLC-5/40, -40L: 000 - 157 PLC-5/60, -5/60L: 000 - 237
IOT	Immediate Output	I/O word	immediate, integer PLC-5/10, 12, 15, 25, 30: 000-077 PLC-5/40, -40L: 000 - 157 PLC-5/60, -5/60L: 000 - 237
JMP	Jump	label number	immediate PLC-5/10, -5/12, -5/15, -5/25: 0-31 PLC-5/30, -5/40, -5/60: 0-255
JSR	Jump to Subroutine	ladder program number	immediate (2 - 999)
		input parameter list	immediate, integer, float
		return parameter list	integer float
LAB	SFC label	filenumber	immediate PLC-5/10, -5/12, -5/15, -5/25: 0-31 PLC-5/30, -5/40, -5/60: 0-255
LBL	LBL (ladder program label)	label number	immediate PLC-5/10, -5/12, -5/15, -5/25: 0-31 PLC-5/30, -5/40, -5/60: 0-255

Instruction	Description	Operand	Valid Value
LEQ	Less Than or Equal To	source A	immediate, integer, float
		source B	immediate, integer, float
LES	Less Than	source A	immediate, integer, float
		source B	immediate, integer, float
LFL *	LIFO Load	source operand	immediate, indexed, integer
		LIFO array	integer
		LIFO control	control
		length	1 - 1000
		position	0 - 999
LFU *	LIFO Unload	LIFO array	integer
		destination	indexed, integer
		LIFO control	control
		length	1 - 1000
		position	0 - 999
LIM	Limit	low limit	immediate, integer, float
		test	immediate, integer, float
		high limit	immediate, integer, float
LN *	Natural Log	source	immediate, integer, float
		destination	immediate, float
LOG *	Log to the Base 10	source	immediate, integer, float
		destination	immediate, float
MCR	Master Control Relay		
MEQ	Mask Compare Equal To	source operand	immediate, integer
		source mask	immediate, integer
		compare operand	immediate, integer
MOV	Move	source	immediate, integer, float
		destination	integer, float
MSG	Message	control block	message, integer
MUL	Multiply	source A	immediate, integer, float
		source B	immediate, integer, float
		destination	integer, float

Instruction	Description	Operand	Valid Value
MVM	Masked Move	source operand	immediate, integer
		source mask	immediate, integer
		destination	integer
NEG	Negate	source	immediate, integer, float
		destination	integer, float
NEQ	Not Equal To	source A	immediate, integer, float
		source B	immediate, integer, float
NOT	Logical NOT	source	immediate, integer
		destination	integer
NSE	SFC next selection branch	only for ASCII import/export	
NSI	SFC next simultaneous branch	only for ASCII import/export	
NXT	Next (FOR Loop)	for label number	immediate, integer (PLC-5/10, -5/12, -5/15, -5/25: 0–31) (PLC-5/30, -5/40, -5/60: 0–255)
ONS	One Shot	source bit	bit
OSR *	One Shot Rising	storage bit	bit
		output bit	immediate (0 – 15)
		output word	integer
OSF *	One Shot Falling	storage bit	bit
		output bit	immediate (0 – 15)
		output word	integer
OR	Logical OR	source A	immediate, integer
		source B	immediate, integer
		destination	integer
OTE	Output Energize	destination bit	bit
OTL	Output Latch	destination bit	bit
OTU	Output Unlatch	destination bit	bit
PID	PID	control block	PID, integer
		pv value	immediate, integer
		tieback value	immediate, integer
		cv value	integer

Instruction	Description	Operand	Valid Value
RAD *	Radian (convert degrees to radians)	source	immediate, float (in degrees)
		destination	immediate, float (in radians)
REF	SFC reference (see LAB)	label number	immediate (0 - 255)
RES	Timer/Counter Reset		timer, counter, control
RET	Return	return parameter list	immediate, integer, float
RTO	Retentive Timer On	timer	timer
		time base	immediate (0.01, 1.0)
		PRE	0 - 32767
		ACC	0 - 32767
SBR	Subroutine	input parameter list	integer, float
SDZ	start of delete zone, unassembled edits	only in ASCII export files	
SEL	SFC selection branch	only for ASCII import/export	
SIM	SFC simultaneous branch	only for ASCII import	
SIN *	Sine	source	immediate, float (in radians)
		destination	immediate, float (in radians)
SIZ	start of insert zone, unassembled edits	only in ASCII export files	
SOC	start of compression	only for ASCII import/export	
SOP	SFC start of program	only for ASCII import/export	
SOR	start of rung	only for ASCII import/export	
SRT *	Sort	sort file	integer, float
		file control	control
		length	1 - 1000
		position	0 - 999
SQI	Sequencer Input	file	integer
		mask	immediate, indexed, integer
		source	immediate, indexed, integer
		control	control
		length	1 - 1000
		position	0 - 999

Instruction	Description	Operand	Valid Value
SQL	Sequencer Load	file	integer
		source	immediate, indexed, integer
		control	control
		length	1 - 1000
		position	0 - 999
SQO	Sequencer Output	file	integer
		destination mask	immediate, indexed, integer
		destination	indexed, integer
		control	control
		length	1 - 1000
		position	0 - 999
SQR	Square Root	source	immediate, integer, float
		destination	integer, float
SRZ	start of replace zone, unassembled edits	only in ASCII export files	
STP	SFC step (PLC-5/10, -5/12, -5/15, -5/25)	file number	2 - 999
STP *	SFC step (PLC-5/30, -5/40, -5/60)	step timer file number	2 - 999
		time base	immediate (0.01, 1.0)
		qualifier	N, S, R, L, D, P1, P0, SL, SD, DS
		action number (from ACT)	immediate
		timer file number	timer
		time base	immediate (0.01, 1.0)
STD *	Standard Deviation	standard deviation file	integer, float
		destination	integer, float
		file control	control
		length	1 - 1000
		position	0 - 999
SUB	Subtract	source A	immediate, integer, float
		source B	immediate, integer, float
		destination	integer, float

Instruction	Description	Operand	Valid Value
TAN *	Tangent	source	immediate, float (in radians)
		destination	immediate, float (in radians)
TID *	Token ID	token ID number (must be unique per SFC file)	immediate
TOD	To BCD	source	immediate, integer
		destination	integer
TOF	Timer Off Delay	timer	timer
		time base	immediate (0.01, 1.0)
		PRE	0 - 32767
		ACC	0 - 32767
TON	Timer On Delay	timer	timer
		time base	immediate (0.01, 1.0)
		PRE	0 - 32767
		ACC	0 - 32767
TND	Temporary End		
TRC	SFC transition	file number	2 - 999
UID *	User Interrupt Disable		
UIE *	User Interrupt Enable		
XIC	Examine On	source bit	bit
XIO	Examine Off	source bit	bit
XOR	Exclusive Or	source A	immediate, integer
		source B	immediate, integer
		destination	integer
XPY *	X to the Power of Y	source A	immediate, integer
		source B	immediate, integer
		destination	immediate, integer

Using This Index

If You See Entry:	Refer to This Module:
Config/Maint	Software Configuration and Maintenance
Prog/Files	Programming – Working with Files section
Prog/SFC	Programming – Entering and Editing SFCs section
Prog/Ladder	Programming – Entering and Editing Ladder Logic section
Instruction	Instruction Set Reference
I/OConfig	I/O Configuration Software

Numbers

1771-IFE, configuring I/O: *I/OConfig* 2-3
1771-IL, configuring I/O: *I/OConfig* 2-4
1771-IR, configuring I/O: *I/OConfig* 2-6
1771-IXE, configuring I/O: *I/OConfig* 2-9
1771-OFE, configuring I/O: *I/OConfig* 2-12
1771-VHSC, configuring I/O: *I/OConfig* 3-1
1784-KL
  configuring: *Config/Maint* 5-3
  DH II: *Config/Maint* 5-3
  DH+: *Config/Maint* 5-3
  DH+ routing: *Config/Maint* 5-3
1784-KT, addressing: *Config/Maint* 5-10
1784-KT/KT2
  configuring: *Config/Maint* 5-7
  DH II: *Config/Maint* 5-7
  DH+: *Config/Maint* 5-7
  DH+ routing: *Config/Maint* 5-7
1784-KT2, addressing: *Config/Maint* 5-11
1784-KTK1
  addressing: *Config/Maint* 5-12
  configuring: *Config/Maint* 5-12
  DH+: *Config/Maint* 5-12
  using 6122: *Config/Maint* 5-14
  using IBM XT: *Config/Maint* 5-14
1784-T45
  considerations: *Config/Maint* 1-2
  using a modem: *Config/Maint* 5-22
6122, using 1784-KTK1: *Config/Maint* 5-14
6171-IDH
  COM1, COM2: *Config/Maint* 1-3
  IRQ settings: *Config/Maint* 1-3
  using with the software: *Config/Maint* 1-2
6174-DMB9: *Config/Maint* 1-1

A

ABL instruction: *Instruction* 17-3
ABMENU.EXE: *Config/Maint* 2-2
ACB instruction: *Instruction* 17-4
ACI instruction: *Instruction* 17-5
ACN instruction: *Instruction* 17-5
active node list. *See* status information, active node
adapter mode
  configuring channels 1A-2B: *Config/Maint* 12-26
  monitoring status: *Config/Maint* 13-16
ADD instruction: *Instruction* 4-10
Addition instruction, ADD: *Instruction* 4-10
addressing
  data files: *Config/Maint* 8-1, 8-7
  frequently used files: *Config/Maint* 8-4
  I/O image: *Config/Maint* 8-12
  I/O status file: *Config/Maint* 10-8
  indexed: *Config/Maint* 8-14
  indirect: *Config/Maint* 8-13
  ladder editor display: *Prog/Ladder* 2-4
  logical: *Config/Maint* 8-7
  mnemonics: *Config/Maint* 8-9
  symbolic: *Config/Maint* 8-15
AEX instruction: *Instruction* 17-6
AFI instruction: *Instruction* 13-14
AHL instruction: *Instruction* 17-7
AIC instruction: *Instruction* 17-8
Always False instruction: *Instruction* 13-14
AND instruction: *Instruction* 5-2

AND Operation instruction, AND: *Instruction* 5-2
ANSI.SYS: *Config/Maint* 1-2
application window, ControlView. *See* ControlView
archive files
  See also processor memory
  .AF5 files: *Prog/Files* 3-5, Files A-1
ARD instruction: *Instruction* 17-8
arithmetic flags. *See* status information, arithmetic
ARL instruction: *Instruction* 17-10
ASC instruction: *Instruction* 17-12
ASCII
  ABL: *Instruction* 17-3
  ACB: *Instruction* 17-4
  ACI: *Instruction* 17-5
  ACN: *Instruction* 17-5
  AEX: *Instruction* 17-6
  AHL: *Instruction* 17-7
  AIC: *Instruction* 17-8
  ARD: *Instruction* 17-8
  ARL: *Instruction* 17-10
  ASC: *Instruction* 17-12
  ASR: *Instruction* 17-13
  AWA: *Instruction* 17-13
  AWT: *Instruction* 17-15
  configuring serial port: *Config/Maint* 12-15
  documenting file: *Prog/Files* 7-12
  import/export: *Prog/Files* 6-1
  status: *Config/Maint* 13-7
ASCII instructions, strings: *Instruction* 17-2
ASCII Integer to String instruction: *Instruction* 17-8
ASCII Read Characters instruction: *Instruction* 17-8
ASCII Read Line instruction: *Instruction* 17-10
ASCII Set Handshake Lines instruction: *Instruction* 17-7
ASCII String Compare instruction: *Instruction* 17-13
ASCII String Concatenate instruction: *Instruction* 17-5
ASCII String Extract instruction: *Instruction* 17-6
ASCII String Search instruction: *Instruction* 17-12
ASCII String to Integer instruction: *Instruction* 17-5
ASCII Write Append instruction: *Instruction* 17-13
ASCII Write instruction: *Instruction* 17-15
ASR instruction: *Instruction* 17-13
assisted search: *Prog/Ladder* 4-1
auto document: *Prog/Ladder* 2-6
AUTOEXEC.BAT: *Config/Maint* 1-2, 1-7, 1-11, 2-2
automatic configuration: *Config/Maint* 12-25, 12-32
AWA instruction: *Instruction* 17-13
AWT instruction: *Instruction* 17-15

B

baud rate
  using a 1784-T45: *Config/Maint* 5-16
  using a 6171-IDH: *Config/Maint* 1-3
  using an IBM XT: *Config/Maint* 5-16
bit, data storage: *Config/Maint* 8-2
Bit Distribute instruction, BTD: *Instruction* 7-1
Bit Shift Left (BSL) instruction: *Instruction* 11-1
Bit Shift Right (BSR) instruction: *Instruction* 11-1
block transfer
  BTR instruction: *Instruction* 15-2
  BTW instruction: *Instruction* 15-2
  direct communication mode: *Instruction* 15-1
  I/O scan mode: *Instruction* 15-1
  instructions: *Instruction* 15-1
  programming examples: *Instruction* 15-14
  timing: *Instruction* 15-12, 15-13
Block Transfer Read instruction, BTR: *Instruction* 15-2

Config/Maint = Software Configuration and Maintenance
I/OConfig = I/O Configuration Software
Instruction = Instruction Set Reference
Prog/Files = Programming, Files section
Prog/SFC = Programming, SFCs section
Prog/Ladder = Programming, Ladder section

Block Transfer Write instruction, BTW: *Instruction* 15-2
branches: *Prog/Ladder* 1-5
  entering multiple: *Prog/Ladder* 3-16
  extending: *Prog/Ladder* 3-16
  ladder editor
    adding: *Prog/Ladder* 3-14
    entering: *Prog/Ladder* 3-13
    nested: *Prog/Ladder* 3-18
Break (BRK) instruction: *Instruction* 13-5
BRK instruction: *Instruction* 13-5
BSL instruction: *Instruction* 11-1
BSR instruction: *Instruction* 11-1
BTD instruction: *Instruction* 7-1
BTR instruction: *Instruction* 15-2
BTR, BTW instruction, monitoring: *Config/Maint* 15-5
BTW instruction: *Instruction* 15-2
building, SFCs: *Prog/SFC* 2-4, 3-11
building blocks
  path: *Prog/SFC* 1-2
  selection branch: *Prog/SFC* 1-3
  SFC: *Prog/SFC* 1-1
  simultaneous branch: *Prog/SFC* 1-4
  step: *Prog/SFC* 1-1
  transition: *Prog/SFC* 1-2

C

CAR utility: *Instruction* 18-1
  restoring files: *Prog/Files* 3-7
changing, passwords: *Config/Maint* 14-3
channel, privileges: *Config/Maint* 14-8
channel 0
  configuration: *Config/Maint* 12-5
  DF1 master status: *Config/Maint* 13-5
  DF1 point-to-point status: *Config/Maint* 13-4
  DF1 slave status: *Config/Maint* 13-5
  interpreting status: *Config/Maint* 13-4
  user mode (ASCII): *Config/Maint* 13-7
channel 1A-2B, monitoring
  Data Highway Plus: *Config/Maint* 13-9
  remote I/O adapter: *Config/Maint* 13-16
  remote I/O scanner: *Config/Maint* 13-12
channel 2, monitoring, expanded local I/O: *Config/Maint* 13-17
channel configuration
  1A: *Config/Maint* 12-19
  1B: *Config/Maint* 12-19
  2A: *Config/Maint* 12-19
  2B: *Config/Maint* 12-19
  adapter, remote I/O: *Config/Maint* 12-26
  channel 0: *Config/Maint* 12-5
  Data Highway Plus: *Config/Maint* 12-20
  expanded local I/O: *Config/Maint* 12-29
  monitoring: *Config/Maint* 13-1
  saving edits: *Config/Maint* 12-5
  scanner, remote I/O: *Config/Maint* 12-23
  unused mode: *Config/Maint* 12-28
channel edits, saving: *Config/Maint* 12-5
channel functions, default: *Config/Maint* 12-1
Channel Overview screen: *Config/Maint* 12-3
channel status
  interpreting for channel 0: *Config/Maint* 13-4
  interpreting for Data Highway Plus: *Config/Maint* 13-9
  interpreting for expanded local I/O: *Config/Maint* 13-17
  interpreting for remote I/O adapter: *Config/Maint* 13-16
  interpreting for remote I/O scanner: *Config/Maint* 13-12
classes, privileges: *Config/Maint* 14-4
Clear instruction, CLR: *Instruction* 4-14
clearing, memory: *Config/Maint* 3-11
clearing faults: *Config/Maint* 13-15, 13-20
CLR instruction: *Instruction* 4-14
CMP, instruction: *Instruction* 3-2
collisions, importing documentation: *Prog/Files* 6-5
colors. *See* Programming Terminal, configuring
COM1, COM2
  communications: *Config/Maint* 5-15

setting the baud rate: *Config/Maint* 1-3
comments
address: *Prog/Ladder* 5-10
import/export. *See* importing, documentation
instruction: *Prog/Ladder* 5-11
ladder editor display: *Prog/Ladder* 2-4
rung: *Prog/Ladder* 5-11
SFC: *Prog/SFC* 3-9
communication mode change
characters used: *Config/Maint* 12-18
configuring serial port: *Config/Maint* 12-18
communication scan time, configuring: *Config/Maint* 10-5
communication time slice. *See* configuring, communication scan time
communications
boards: *Config/Maint* 5-1
DH+: *Config/Maint* 5-14
modem: *Config/Maint* 5-19
serial: *Config/Maint* 5-15
using DH+: *Config/Maint* 7-1
compare
EQU: *Instruction* 3-4
expression: *Instruction* 3-2
GEQ: *Instruction* 3-5
GRT: *Instruction* 3-5
instructions: *Instruction* 3-2
length of expressions: *Instruction* 3-3
LEQ: *Instruction* 3-5
LES: *Instruction* 3-6
NEQ: *Instruction* 3-9
comparing
data table entries: *Prog/Files* 4-1
force table entries: *Prog/Files* 4-1
ladder program: *Prog/Files* 4-1
modes: *Prog/Files* 4-2
options: *Prog/Files* 4-5
processor memory: *Prog/Files* 4-1
program directory: *Prog/Files* 4-1
report: *Prog/Files* 4-4
compute
ADD: *Instruction* 4-10
CLR: *Instruction* 4-14
CPT: *Instruction* 4-4
DIV: *Instruction* 4-15
expression: *Instruction* 4-4
functions: *Instruction* 4-8
length of expressions: *Instruction* 4-6
MUL: *Instruction* 4-17
NEG: *Instruction* 4-10, 4-11, 4-12, 4-15, 4-16, 4-17, 4-18, 4-24, 4-25, 6-3, 6-4
order of operation: *Instruction* 4-6
SQR: *Instruction* 4-19
SUB: *Instruction* 4-24
Compute instruction, CPT: *Instruction* 4-4
computers, supported by software: *Config/Maint* 1-4
concepts, data storage: *Config/Maint* 8-1
CONFIG.SYS
description: *Config/Maint* 1-7
FILES and BUFFERS: *Config/Maint* 1-2
setting: *Config/Maint* 1-1
configurable application routine. *See* CAR utility
configuration
files: *Config/Maint* 2-5
I/O. *See* I/O configuration
online: *Config/Maint* 5-1
system: *Config/Maint* 4-1
configuring
1771-VHSC: *I/OConfig* 3-1
1784-KL: *Config/Maint* 5-3
1784-KT/KT2: *Config/Maint* 5-7
1784-KTK1: *Config/Maint* 5-12
adapter image file: *Config/Maint* 10-11
analog I/O modules: *I/OConfig* 2-1
channel 0
ASCII (user mode): *Config/Maint* 12-15
communication mode change: *Config/Maint* 12-18
DF1 master: *Config/Maint* 12-11
DF1 point-to-point: *Config/Maint* 12-6
DF1 slave: *Config/Maint* 12-8

Config/Maint = Software Configuration and Maintenance
I/OConfig = I/O Configuration Software
Instruction = Instruction Set Reference
Prog/Files = Programming, Files section
Prog/SFC = Programming, SFCs section
Prog/Ladder = Programming, Ladder section

channels 1A-2B
as Data Highway Plus: *Config/Maint* 12-20
as remote I/O adapter: *Config/Maint* 12-26
as remote I/O scanner: *Config/Maint* 12-23
colors: *Config/Maint* 4-3
communication scan time: *Config/Maint* 10-5
directory paths: *Config/Maint* 4-5
display: *Config/Maint* 4-1
documentation, export utility: *Prog/Files* 6-10
fault routine: *Config/Maint* 10-6
I/O, automatic: *Config/Maint* 11-24
I/O status file: *Config/Maint* 10-7
import documentation: *Prog/Files* 6-4
intelligent I/O modules. *See* I/O configuration
ladder editor: *Prog/Ladder* 2-4
link mode: *Config/Maint* 4-4
local rack addres: *Config/Maint* 10-5
main control programs: *Config/Maint* 10-14
offline: *Config/Maint* 6-1
PII: *Config/Maint* 10-12
printer: *Config/Maint* 4-5, 19-16
processor: *Config/Maint* 10-1
processor channels: *Config/Maint* 12-1
processor memory file, import: *Prog/Files* 7-4
processor memory files, export: *Prog/Files* 7-8
programming terminal: *Config/Maint* 4-1
resident local rack address: *Config/Maint* 10-5
serial port: *Config/Maint* 5-15, 12-5
ASCII (user mode): *Config/Maint* 12-15
communication mode change: *Config/Maint* 12-18
DF1 master: *Config/Maint* 12-11
DF1 point-to-point: *Config/Maint* 12-6
DF1 slave: *Config/Maint* 12-8
SFC display: *Prog/SFC* 1-12
SFCs: *Prog/SFC* 5-10
snow control: *Config/Maint* 4-3
startup procedure: *Config/Maint* 10-3
STI: *Config/Maint* 10-13
constants, program: *Config/Maint* 8-17
constructing, rungs: *Prog/Ladder* 1-3
contact histogram
generating: *Config/Maint* 18-1
printing: *Config/Maint* 18-4
using masks: *Config/Maint* 18-3
control bits. *See* user control bits
ControlView
6200 requirements: *Config/Maint* A-1
alarm messages in 6200: *Config/Maint* A-7
configuring colors: *Config/Maint* A-2
configuring communications: *Config/Maint* A-4
running 6200: *Config/Maint* A-1
convergent, scan time: *Instruction* B-13
conversion
BCD: *Instruction* 6-2
FRD: *Instruction* 6-2
Convert from BCD instruction, FRD: *Instruction* 6-2
Convert to BCD instruction, TOD: *Instruction* 6-2
converting machine statements, ladder logic: *Prog/Ladder* 1-1
COP instruction: *Instruction* 9-17
copy rungs: *Prog/Ladder* 4-15
using disk files: *Prog/Ladder* 4-17
using the buffer: *Prog/Ladder* 4-16
copying, files: *Prog/Files* 5-3, 5-5
Count Down instruction: *Instruction* 2-14
Count Up instruction: *Instruction* 2-12
counter
CTD: *Instruction* 2-14
CTU: *Instruction* 2-12
RES: *Instruction* 2-18
counters
instructions: *Instruction* 2-11

rack re-scans: *Config/Maint* 13-15
CPT instruction: *Instruction* 4-4
creating
offline files: *Config/Maint* 6-3
program files: *Prog/Files* 2-3
reports: *Config/Maint* 19-3
SFCs: *Prog/SFC* 2-1, 3-1
cross reference
display: *Prog/Ladder* 2-4
displaying: *Prog/Ladder* 5-15
report: *Config/Maint* 19-8, 19-9
CTD instruction: *Instruction* 2-14
CTU instruction: *Instruction* 2-12
custom application routine. *See* CAR utility
cut rungs: *Prog/Ladder* 4-15
using disk files: *Prog/Ladder* 4-17
using the buffer: *Prog/Ladder* 4-16

D

data monitor
message: *Config/Maint* 15-6, 15-9
string data: *Config/Maint* 15-11
data block: *Config/Maint* 8-3
organizing: *Config/Maint* 8-3
user–defined: *Config/Maint* 8-3
data file
addressing: *Config/Maint* 8-1
creating: *Config/Maint* 9-1
default: *Config/Maint* 8-5
default formats: *Config/Maint* 15-3
deleting: *Config/Maint* 9-3
extending: *Config/Maint* 9-3
manipulating: *Instruction* 8-3
memory used: *Config/Maint* 8-5
organizing: *Config/Maint* 8-1, 8-3
privileges: *Config/Maint* 14-13
report: *Config/Maint* 19-9
size: *Config/Maint* 8-5
types of: *Config/Maint* 8-5
data files
types of: *Config/Maint* 8-7
types of addressing: *Config/Maint* 8-7
Data Highway Plus
active nodes: *Config/Maint* 11-12
configuring channels 1A-2B: *Config/Maint* 12-20
interpreting status: *Config/Maint* 13-9
monitoring status: *Config/Maint* 13-9
data monitor: *Config/Maint* 15-1
block transfer: *Config/Maint* 15-5
changing the radix: *Config/Maint* 15-2
entering data: *Config/Maint* 15-4
specific address: *Config/Maint* 15-4
data storage
bit: *Config/Maint* 8-2
concepts: *Config/Maint* 8-1
data block: *Config/Maint* 8-3
files: *Config/Maint* 8-2
I/O image files: *Instruction* 1-1
sub-member: *Config/Maint* 8-3
type: *Config/Maint* 8-2
user–defined: *Config/Maint* 8-3
words: *Config/Maint* 8-2
data table, report: *Config/Maint* 19-9
data table files, default formats: *Config/Maint* 15-3
Data Transitional instruction, DTR: *Instruction* 10-6
database editor
address comment: *Prog/Ladder* 5-10
displaying cross reference: *Prog/Ladder* 5-15
instruction comment: *Prog/Ladder* 5-11
modes: *Prog/Ladder* 5-11
replacing: *Prog/Ladder* 5-14
rung comment: *Prog/Ladder* 5-11
search order: *Prog/Ladder* 5-14
search types: *Prog/Ladder* 5-13
searching: *Prog/Ladder* 5-12
using: *Prog/Ladder* 5-8
DDT instruction: *Instruction* 10-1
default
channel configuration: *Config/Maint* 12-1
data table: *Config/Maint* 8-5

Config/Maint = Software Configuration and Maintenance
I/OConfig = I/O Configuration Software
Instruction = Instruction Set Reference
Prog/Files = Programming, Files section
Prog/SFC = Programming, SFCs section
Prog/Ladder = Programming, Ladder section

program file: *Prog/Ladder* 2-2
defining, privilege class: *Config/Maint* 14-4
deleting
files: *Prog/Files* 5-9
I/O module configuration: *I/OConfig* 1-6
derivative smoothing: *Instruction* 14-3
DFA instruction: *Instruction* 18-1
DH II, serial port configuring: *Config/Maint* 5-15
DH+
communications: *Config/Maint* 7-1
between DH+ networks: *Config/Maint* 5-14
serial port configuring: *Config/Maint* 5-15
DH+ Network Menu: *Config/Maint* 2-3, 7-5
DH+ station. *See* status information, DH+
diagnostic
DDT: *Instruction* 10-1
DTR: *Instruction* 10-6
FBC: *Instruction* 10-1
parameters: *Instruction* 10-3, 10-7
search mode: *Instruction* 10-2
status: *Instruction* 10-4
Diagnostic Detect instruction, DDT: *Instruction* 10-1
Diagnostic Fault Annunciator Instruction: *Instruction* 18-1
diagnostic instructions: *Instruction* 10-1
diagnostics: *Config/Maint* 7-6
digital I/O, example instructions: *Prog/Ladder* 1-3
direct communication, block transfer: *Instruction* 15-1
directory paths, defining: *Config/Maint* 4-5
disabling, I/O forces: *Config/Maint* 17-5
disk files: *Prog/Files* 1-1
display
configuring: *Config/Maint* 4-1
ladder editor: *Prog/Ladder* 2-4
SFC: *Prog/SFC* 1-12
DIV instruction: *Instruction* 4-15
divergent, scan time: *Instruction* B-13
Divide instruction, DIV: *Instruction* 4-15
documentation
exporting: *Prog/Files* 6-8
importing: *Prog/Files* 6-2
documenting, ASCII file: *Prog/Files* 7-12
documenting program files: *Prog/Ladder* 5-1
auto document: *Prog/Ladder* 2-6
editing keys: *Prog/Ladder* 5-5
entering comments: *Prog/Ladder* 5-3
entering symbols: *Prog/Ladder* 5-7
guidelines: *Prog/Ladder* 5-4
using the database editor: *Prog/Ladder* 5-8
DOS: *Config/Maint* 1-1
downloading, SDS CAR: *Instruction* 18-2, 18-5
drawing an SFC: *Prog/SFC* 1-1, 1-7
DTR instruction: *Instruction* 10-6
duplicate check, ladder editor display: *Prog/Ladder* 2-4

E

edit mode, SFCs: *Prog/SFC* 1-14
editing
module configurations: *I/OConfig* 1-6
SFC: *Prog/SFC* 5-1
SFC branches: *Prog/SFC* 4-1
edits, testing: *Prog/Ladder* 4-20
modes: *Prog/Ladder* 4-21
EEPROM, writing to: *Prog/Files* 3-8
element manipulation
LIM: *Instruction* 3-6, 4-12, 4-20, 4-21
MEQ: *Instruction* 3-7
MOV: *Instruction* 7-3
MVM: *Instruction* 7-3
EMM.SYS: *Config/Maint* 1-2
enabling, I/O forces: *Config/Maint* 17-5
engineering units, scaling: *Instruction* 14-4
entering
SDS instruction: *Instruction* 18-3
step actions, transitions: *Prog/SFC* 3-3
step/transition pairs: *Prog/SFC* 2-3
EOT instruction, using: *Prog/SFC* 2-3, 3-3

EQU instruction: *Instruction* 3-4
Equal To instruction: *Instruction* 3-4
ERR: *Prog/Ladder* 3-3
Examine Off instruction: *Instruction* 1-3
Examine On instruction: *Instruction* 1-2
example
  indexed addressing: *Config/Maint* 8-15
  privilege classes: *Config/Maint* 14-4
executable, starting the software: *Config/Maint* 2-3
exiting the software: *Config/Maint* 2-6
expanded local I/O
  concepts: *Config/Maint* 12-29
  configuration table: *Config/Maint* 13-19
  configuring channel 2: *Config/Maint* 12-29
  interpreting status: *Config/Maint* 13-17
  monitoring: *Config/Maint* 13-19
  monitoring status: *Config/Maint* 13-17
expanded memory: *Config/Maint* 1-3
exporting
  creating A-B ASCII file: *Prog/Files* 6-14, 7-9
  documentation: *Prog/Files* 6-8
    configuring: *Prog/Files* 6-10
  documentation files: *Prog/Files* 6-1
  examining results: *Prog/Files* 7-9
  processor memory files: *Prog/Files* 7-6
    configuring: *Prog/Files* 7-8
    documenting: *Prog/Files* 7-12
  valid operands: *Instruction* C-1
expression, determining the length of: *Instruction* 3-3, 4-6
extended memory: *Config/Maint* 1-3
extending, data files: *Config/Maint* 9-3
extensions, reports: *Config/Maint* 19-13
external format processor memory file: *Prog/Files* B-1

F

FAL logical instruction: *Instruction* 9-11
fault routine, configuring: *Config/Maint* 10-6
faults
  clearing: *Config/Maint* 11-6, 11-7, 16-1
  clearing major: *Config/Maint* 16-2
  clearing minor: *Config/Maint* 16-6
  codes: *Config/Maint* 11-5, 11-7, 16-4
  description: *Config/Maint* 16-3
  major: *Config/Maint* 11-6
    clearing: *Config/Maint* 16-2
    codes: *Config/Maint* 16-4
    types of: *Config/Maint* 16-3
  major and minor: *Config/Maint* 11-5
  minor: *Config/Maint* 11-7
    clearing: *Config/Maint* 16-6
    types of: *Config/Maint* 16-7
  minor fault word 1 description: *Config/Maint* 16-7
  minor fault word 2 description: *Config/Maint* 16-8
  remote I/O rack: *Config/Maint* 11-6
  status information: *Config/Maint* 11-5
  where faulted: *Config/Maint* 11-7
FBC instruction: *Instruction* 10-1
FFL instruction: *Instruction* 11-5
FFU instruction: *Instruction* 11-5
FIFO Load (FFL) instruction: *Instruction* 11-5
FIFO Unload (FFU) instruction: *Instruction* 11-5
file
  arithmetic operations: *Instruction* 9-5
  copy operations: *Instruction* 9-3
  functions: *Instruction* 9-12
  instruction, FLL: *Instruction* 9-18
  instructions, COP: *Instruction* 9-17
  logical operations: *Instruction* 9-11
  operation modes: *Instruction* 8-4
  search and compare operations: *Instruction* 9-15
File Arithmetic and Logic instruction, FAL: *Instruction* 9-1
File Bit Comparison instruction, FBC: *Instruction* 10-1
file concepts
  control structure: *Instruction* 8-2

manipulating data: *Instruction* 8-3
modes of operation: *Instruction* 8-4
parameters: *Instruction* 8-1
file convert utility: *Prog/Files* 3-5, Files A-1
File Copy instruction, COP: *Instruction* 9-17
File Fill instruction, FLL: *Instruction* 9-18
file instructions, logical: *Instruction* 9-11
File Search and Compare instruction, FSC: *Instruction* 9-13
files
archive. *See* Processor Memory Files
copying
same directory: *Prog/Files* 5-3
to/from floppy: *Prog/Files* 5-5
creating offline: *Config/Maint* 6-3
data storage: *Config/Maint* 8-2
deleting: *Config/Maint* 9-3; *Prog/Files* 5-9
disk: *Prog/Files* 1-1
documenting: *Prog/Ladder* 5-1
editing keys: *Prog/Ladder* 5-5
guidelines: *Prog/Ladder* 5-4
entering symbols: *Prog/Ladder* 5-7
extending: *Config/Maint* 9-3
extensions: *Prog/Files* 1-2
frequently used: *Config/Maint* 8-4
merging documentation: *Prog/Files* 5-10
merging processor memory: *Prog/Files* 5-14
offline, selecting: *Config/Maint* 6-4
passwords: *Config/Maint* 14-1
privileges: *Config/Maint* 14-11
processor: *Config/Maint* 6-3; *Prog/Files* 1-3
processor memory, naming: *Prog/Files* 2-5
program: *Prog/Files* 2-1
naming: *Prog/Files* 2-5
renaming: *Prog/Files* 5-1
reports: *Config/Maint* 19-13
restoring: *Prog/Files* 3-4
saving: *Prog/Files* 3-1
types of: *Prog/Files* 1-1
understanding: *Prog/Files* 1-1
user configuration: *Config/Maint* 2-5
utilities: *Prog/Files* 5-1
FLL instruction: *Instruction* 9-18
floating point, valid value range: *Instruction* C-1
For (FOR) instruction: *Instruction* 13-5
FOR instruction: *Instruction* 13-5
forcing, SFC transitions: *Config/Maint* 17-6; *Prog/SFC* 5-7
forcing I/O
disabling: *Config/Maint* 17-5
enabling: *Config/Maint* 17-5
from Force Monitor: *Config/Maint* 17-3
from ladder editor: *Config/Maint* 17-1
on/off: *Config/Maint* 17-4
removing forces: *Config/Maint* 17-5
FRD instruction: *Instruction* 6-2
function keys: *Config/Maint* 3-4

G

gain constants: *Instruction* 14-3
gapping: *Config/Maint* 8-4
generating
contact histogram: *Config/Maint* 18-1
reports: *Config/Maint* 19-1
GEQ instruction: *Instruction* 3-5
global control bits: *Config/Maint* 11-20
global inhibit bits, clearing: *Config/Maint* 12-25, 12-32
global status bits: *Config/Maint* 11-18
global status flags file: *Config/Maint* 12-21
GO TO
definition: *Prog/SFC* 1-8
entering: *Prog/SFC* 4-11, 4-12
placing: *Prog/SFC* 4-9
rules: *Prog/SFC* 1-9
using: *Prog/SFC* 4-7
Greater Than instruction: *Instruction* 3-5
Greater Than or Equal To instruction: *Instruction* 3-5
GRT instruction: *Instruction* 3-5

H

help: *Config/Maint* 3-5

histogram. *See* contact histogram
history, of data table values. *See* contact histogram
hot keys
  available throughout software: *Config/Maint* 3-3
  entering comments: *Prog/Ladder* 5-6
  moving between screens: *Config/Maint* 3-4
  SFC Editor: *Prog/SFC* 1-15

I
I/O, automatic configuration: *Config/Maint* 11-24
I/O configuration: *I/OConfig* 1-1
  accepting edits: *I/OConfig* 2-2, 1-8, 3-2
  availability: *I/OConfig* 1-1
  configuring 1771-IFE: *I/OConfig* 2-3
  configuring 1771-IL: *I/OConfig* 2-4
  configuring 1771-IR: *I/OConfig* 2-6
  configuring 1771-IXE: *I/OConfig* 2-9
  configuring 1771-OFE: *I/OConfig* 2-12
  configuring 1771-VHSC: *I/OConfig* 3-1
  configuring analog: *I/OConfig* 2-1
  coverting: *I/OConfig* A-1
  deleting module: *I/OConfig* 1-6
  documenting: *I/OConfig* 1-3
  editing, modules: *I/OConfig* 1-6
  I/O module system overview: *I/OConfig* 1-2
  initiating BTW: *I/OConfig* 2-2, 1-8, 3-2
  modifying module: *I/OConfig* 1-5
  module configurations: *I/OConfig* 1-6
  monitoring modules: *I/OConfig* 1-9
  selecting module: *I/OConfig* 1-4
  starting: *I/OConfig* 1-2
  utility: *I/OConfig* 1-1
I/O image, address: *Config/Maint* 8-12
I/O image files: *Instruction* 1-1
I/O modules
  deleting: *I/OConfig* 1-6
  documenting config info: *I/OConfig* 1-3
  editing, configurations: *I/OConfig* 1-6
  modifying config info: *I/OConfig* 1-5
  monitoring: *I/OConfig* 1-9
I/O rack, configuration table: *Config/Maint* 13-14
I/O scan mode, block transfer: *Instruction* 15-1
I/O status: *Config/Maint* 11-15
  monitoring: *Config/Maint* 13-14
  rack configuration tables: *Config/Maint* 11-21
I/O status file
  addressing: *Config/Maint* 10-8
  configuring: *Config/Maint* 10-7
IBM XT, using 1784-KTK1: *Config/Maint* 5-14
IIN instruction: *Instruction* 1-5, 1-6
Immediate Input instruction: *Instruction* 1-5, 1-6
importing
  comments/symbols. *See* importing, documentation
  creating A-B ASCII file: *Prog/Files* 6-14, 7-9
  documentation: *Prog/Files* 6-2
    configuring: *Prog/Files* 6-4
    data collisions: *Prog/Files* 6-5
    examing results: *Prog/Files* 6-7
    utility screen: *Prog/Files* 6-4
  documentation files: *Prog/Files* 6-1
  documenting ASCII: *Prog/Files* 6-18
  examining results: *Prog/Files* 7-5
  processor memory files: *Prog/Files* 7-1
    configuring: *Prog/Files* 7-4
    documenting: *Prog/Files* 7-12
  valid operands: *Instruction* C-1
importing and exporting
  .X5 to/from .AF5: *Prog/Files* B-1
  documentation files: *Prog/Files* 6-1
incremental mode: *Instruction* 8-8
indexed
  address: *Config/Maint* 8-14
  addressing example: *Config/Maint* 8-15
indexed addressing offset. *See* status information, indexed

indirect, address: *Config/Maint* 8-13
INFO.TXT: *Config/Maint* 1-7
inhibiting expanded local I/O racks: *Config/Maint* 13-20
inhibiting I/O racks: *Config/Maint* 13-15
input keys: *Config/Maint* 3-3
installing
  registration: *Config/Maint* 1-6
  Smart Menu: *Config/Maint* 1-8
  software: *Config/Maint* 1-1, 1-5
  utilities: *Config/Maint* 1-7
  verifying: *Config/Maint* 1-8
instructions
  adding
    command line entry: *Prog/Ladder* 3-6, 3-7
    function key entry: *Prog/Ladder* 3-4
  arranging: *Prog/Ladder* 1-4, 1-6
  ASCII: *Instruction* 17-1
  block transfer: *Instruction* 15-1
  BTR, BTW, monitoring: *Config/Maint* 15-5
  compare: *Instruction* 3-2
  diagnostic: *Instruction* 10-1
  entering: *Prog/Ladder* 3-3
  examples: *Prog/Ladder* 1-3
  memory requirements: *Instruction* A-1
  message: *Instruction* 16-1
  MSG, monitoring: *Config/Maint* 15-6
  operands: *Instruction* C-1
  PID, monitoring: *Config/Maint* 15-9
  program flow: *Instruction* 13-1
  quick reference: *Config/Maint* B-1
  relay–type: *Instruction* 1-1
  sequencer: *Instruction* 12-1
  shift register: *Instruction* 11-1
  string, monitoring: *Config/Maint* 15-11
  timer: *Instruction* 2-1
  timing: *Instruction* A-1
  using instruction entry aid: *Prog/Ladder* 3-7
  valid operands: *Instruction* C-1
internal modem: *Config/Maint* 5-21
IRD marking: *Prog/Ladder* 3-2
IRQ settings: *Config/Maint* 1-3, 5-16
IT executable: *Config/Maint* 2-3
  See also programming software, starting

J

JMP instruction: *Instruction* 13-3
JSR instruction: *Instruction* 13-8
Jump instruction: *Instruction* 13-3
Jump to Subroutine instruction: *Instruction* 13-8

K

keys
  function: *Config/Maint* 3-4
  input: *Config/Maint* 3-3
  ladder editor: *Prog/Ladder* 2-3
  navigational: *Config/Maint* 3-4
keywords, using: *Prog/Files* 6-6

L

Label (LBL) instruction: *Instruction* 13-5
Label instruction: *Instruction* 13-3
labels: *Prog/SFC* 1-8, 4-7
  entering: *Prog/SFC* 4-11, 4-12
  placing: *Prog/SFC* 4-9
  rules: *Prog/SFC* 1-9
ladder editor
  auto document: *Prog/Ladder* 2-6
  branches
    adding: *Prog/Ladder* 3-14
    entering: *Prog/Ladder* 3-13
    extending: *Prog/Ladder* 3-16
    multiple: *Prog/Ladder* 3-16
    nested: *Prog/Ladder* 3-18
  configuring: *Prog/Ladder* 2-4
  copy rungs: *Prog/Ladder* 4-15
    using disk files: *Prog/Ladder* 4-17
    using the buffer: *Prog/Ladder* 4-16
  cut rungs: *Prog/Ladder* 4-15
    using disk files: *Prog/Ladder* 4-17
    using the buffer: *Prog/Ladder* 4-16

documenting program files: *Prog/Ladder* 5-1
editing keys: *Prog/Ladder* 5-5
entering comments: *Prog/Ladder* 5-3
guidelines: *Prog/Ladder* 5-4
entering
instructions: *Prog/Ladder* 3-3
rungs: *Prog/Ladder* 3-1
symbols: *Prog/Ladder* 5-7
forcing I/O: *Config/Maint* 17-1
getting started: *Prog/Ladder* 2-1
IRD marking: *Prog/Ladder* 3-2
moving to rungs: *Prog/Ladder* 2-7
navigational keys: *Prog/Ladder* 2-3
online help: *Config/Maint* 3-5
paste rungs: *Prog/Ladder* 4-15
paste errors: *Prog/Ladder* 4-20
using disk files: *Prog/Ladder* 4-17
using the buffer: *Prog/Ladder* 4-16
viewing paste file: *Prog/Ladder* 4-19
searching: *Prog/Ladder* 4-1
assisted search: *Prog/Ladder* 4-1
configuring: *Prog/Ladder* 4-13
edit zones: *Prog/Ladder* 4-10
instructions/addresses: *Prog/Ladder* 4-4
replacing: *Prog/Ladder* 4-8
standard search: *Prog/Ladder* 4-3
types: *Prog/Ladder* 4-4
undocumented instructions: *Prog/Ladder* 4-12
selecting rungs: *Prog/Ladder* 2-6
status messages: *Prog/Ladder* 2-8
testing edits: *Prog/Ladder* 4-20
modes: *Prog/Ladder* 4-21
using the database editor: *Prog/Ladder* 5-8
ladder logic
branch: *Prog/Ladder* 1-5
entering: *Prog/Ladder* 3-1, 3-3
writing: *Prog/Ladder* 1-3
ladder program
comparing: *Prog/Files* 4-1
report: *Config/Maint* 19-8
LBL instruction: *Instruction* 13-3, 13-5
LEQ instruction: *Instruction* 3-5
LES instruction: *Instruction* 3-6
Less Than instruction: *Instruction* 3-6
Less Than or Equal To instruction: *Instruction* 3-5
LFL instruction: *Instruction* 11-5
LFU instruction: *Instruction* 11-5
LIFO Load (LFL) instruction: *Instruction* 11-5
LIFO Unload (LFU) instruction: *Instruction* 11-5
LIM instruction: *Instruction* 3-6, 4-12, 4-20, 4-21
Limit Test instruction: *Instruction* 3-6, 4-12, 4-20, 4-21
link mode. *See* Programming Terminal, configuring
logic
branch: *Prog/Ladder* 1-5
writing: *Prog/Ladder* 1-3
logical
AND: *Instruction* 5-2
NOT: *Instruction* 5-3
OR: *Instruction* 5-4
XOR: *Instruction* 5-5
logical address: *Config/Maint* 8-7
mnemonic: *Config/Maint* 8-9

M

machine statements, converting to logic: *Prog/Ladder* 1-1
macros: *Prog/SFC* 4-12
assigning comments: *Prog/SFC* 4-14
creating: *Prog/SFC* 4-13
deleting step/tran: *Prog/SFC* 4-14
searching within: *Prog/SFC* 4-15
main control programs: *Prog/Files* 2-2; *Prog/SFC* 1-1
configuring: *Config/Maint* 10-14
main menu, using: *Config/Maint* 3-1
major faults: *Config/Maint* 16-3
manipulating, file data: *Instruction* 8-3

Masked Comparison for Equal instruction: *Instruction* 3-7
Masked Move instruction: *Instruction* 7-3
Master Control Reset instruction: *Instruction* 13-2
MCP. *See* main control programs
MCR instruction: *Instruction* 13-2
memory
  choosing memory size: *Config/Maint* 3-11
  clearing: *Config/Maint* 3-11
  expanded or extended: *Config/Maint* 1-3
  gapping: *Config/Maint* 8-4
  hard disk: *Config/Maint* 1-1
  instruction requirements: *Instruction* A-1
  processor files: *Config/Maint* 6-3
  SFC requirements: *Instruction* B-3
  size, offline: *Config/Maint* 3-11
  usage report: *Config/Maint* 19-9
Memory Map: *Config/Maint* 9-1
MEQ instruction: *Instruction* 3-7
merging
  files
    configuring: *Prog/Files* 5-12
    documentation: *Prog/Files* 5-10
    processor memory: *Prog/Files* 5-14
  rules: *Prog/Files* 5-16
message
  ASCII: *Instruction* 16-3
  communication choices: *Instruction* 16-4
  instruction: *Instruction* 16-1
  ladder editor: *Prog/Ladder* 2-8
minor fault
  word 1 description: *Config/Maint* 16-7
  word 2 description: *Config/Maint* 16-8
mnemonic, addressing: *Config/Maint* 8-9
modem
  string prefixes: *Config/Maint* 5-20
  using: *Config/Maint* 5-19
  using a T45: *Config/Maint* 5-22
  using an internal: *Config/Maint* 5-21
modes
  comparing: *Prog/Files* 4-2
  file operation: *Instruction* 8-4
modifying
  I/O module configuration: *I/OConfig* 1-5
  Smart Menu: *Config/Maint* 1-9
monitor, status information: *Config/Maint* 11-3
monitor mode, SFCs: *Prog/SFC* 1-14
monitoring
  active steps: *Prog/SFC* 5-10
  BTR, BTW instructions: *Config/Maint* 15-5
  changing radix: *Config/Maint* 15-2
  data: *Config/Maint* 15-1
  entering data: *Config/Maint* 15-4
  I/O modules: *I/OConfig* 1-9
  I/O status: *Config/Maint* 11-1
  MSG instructions: *Config/Maint* 15-6
  PID instructions: *Config/Maint* 15-9
  processor channels: *Config/Maint* 13-1
  processor status: *Config/Maint* 11-1
  specific addresses: *Config/Maint* 15-4
  status data, processor: *Config/Maint* 11-1
  string data: *Config/Maint* 15-11
MOV instruction: *Instruction* 7-3
Move instruction, MOV: *Instruction* 7-3
moving, to rungs: *Prog/Ladder* 2-7
MSG instruction: *Instruction* 16-1
  monitoring: *Config/Maint* 15-6
MUL instruction: *Instruction* 4-17
multiple rung display: *Prog/Ladder* 2-4
Multiply instruction, MUL: *Instruction* 4-17
MVM instruction: *Instruction* 7-3

N

naming
  processor memory files: *Prog/Files* 2-5
  program files: *Prog/Files* 2-5
navigational keys: *Config/Maint* 3-4
NEG instruction: *Instruction* 4-10, 4-11, 4-12, 4-15, 4-16, 4-17, 4-18, 4-24, 4-25, 6-3, 6-4
Negate instruction, NEG: *Instruction* 4-10, 4-11, 4-12, 4-15, 4-16, 4-17, 4-18, 4-24, 4-25, 6-3, 6-4
NEQ instruction: *Instruction* 3-9

Next (NXT) instruction: *Instruction* 13-5
node, privileges: *Config/Maint* 14-10
Not Equal To instruction: *Instruction* 3-9
NOT instruction: *Instruction* 5-3
NOT Operation instruction, NOT: *Instruction* 5-3
Number of Char in Buffer instruction: *Instruction* 17-4
NXT instruction: *Instruction* 13-5

O

offline
  configuring: *Config/Maint* 6-1
  creating
    files: *Config/Maint* 6-3
    reports: *Config/Maint* 19-5
  memory size: *Config/Maint* 3-11
  processor series: *Config/Maint* 3-10
  programming: *Config/Maint* 6-1
  selecting files: *Config/Maint* 6-4
offline programming: *Config/Maint* 3-8
  SFC considerations: *Prog/SFC* 2-3, 3-3
One Shot Falling (OSF) instruction: *Instruction* 13-15
One Shot Rising (OSR) instruction: *Instruction* 13-15
One Shot Rising instruction: *Instruction* 13-14
online
  clearing memory: *Config/Maint* 3-11
  communications: *Config/Maint* 5-1
  creating reports: *Config/Maint* 19-3
  programming: *Config/Maint* 3-6
operands, instructions: *Instruction* C-1
OR instruction: *Instruction* 5-4
OR Operation instruction, OR: *Instruction* 5-4
order of operation: *Instruction* 4-6
organizing
  data: *Config/Maint* 8-3
  data files: *Config/Maint* 8-1
  program files: *Prog/Files* 2-1
OSF instruction: *Instruction* 13-15
OSR instruction: *Instruction* 13-14, 13-15
OTE instruction: *Instruction* 1-3
OTL instruction: *Instruction* 1-4
OTU instruction: *Instruction* 1-5
Output Energize instruction: *Instruction* 1-3
Output Latch instruction: *Instruction* 1-4
Output Unlatch instruction: *Instruction* 1-5

P

pairs, step/transition: *Prog/SFC* 2-3
passwords: *Config/Maint* 14-1
  adding to file: *Config/Maint* 14-1
  changing: *Config/Maint* 14-3
  entering: *Config/Maint* 14-14
  removing: *Config/Maint* 14-3
paste rungs: *Prog/Ladder* 4-15
  paste errors: *Prog/Ladder* 4-20
  using disk files: *Prog/Ladder* 4-17
  using the buffer: *Prog/Ladder* 4-16
  viewing paste file: *Prog/Ladder* 4-19
PID
  biasing: *Instruction* 14-8
  equations: *Instruction* 14-2
  examples: *Instruction* 14-25
  instruction: *Instruction* 14-9
  selecting derivative term: *Instruction* 14-5
  setting output alarms: *Instruction* 14-6
  using manual mode: *Instruction* 14-6
  using output limiting: *Instruction* 14-6
PID instruction: *Instruction* 14-1
  monitoring: *Config/Maint* 15-9
PII
  configuration fields: *Config/Maint* 10-12
  configuring: *Config/Maint* 10-12
  status: *Config/Maint* 11-14
placing
  GOTO statements: *Prog/SFC* 4-9
  instructions: *Prog/Ladder* 1-6
  labels: *Prog/SFC* 4-9
PLC-5 processor, choosing series/revision: *Config/Maint* 3-10
PLC-5/VME, status information: *Config/Maint* 11-32

PLC-5/VME status file: *Config/Maint* 10-11
printer, configuring: *Config/Maint* 4-5
printing
  contact histogram: *Config/Maint* 18-4
  reports: *Config/Maint* 19-13
privileges
  assigning to channels: *Config/Maint* 14-8
  assigning to data files: *Config/Maint* 14-13
  assigning to nodes: *Config/Maint* 14-10
  assigning to program files: *Config/Maint* 14-11
  defining classes: *Config/Maint* 14-4
  overview: *Config/Maint* 14-3
  restoring defaults: *Config/Maint* 14-15
  types of: *Config/Maint* 14-3
process control
  biasing: *Instruction* 14-8
  derivative smoothing: *Instruction* 14-3
  equations: *Instruction* 14-2
  gain constants: *Instruction* 14-3
  PID: *Instruction* 14-9
  PID examples: *Instruction* 14-25
  PID instruction: *Instruction* 14-1
  seleting derivative term: *Instruction* 14-5
  setting output alarms: *Instruction* 14-6
  using manual mode: *Instruction* 14-6
  using output limiting: *Instruction* 14-6
processor
  configuring: *Config/Maint* 10-1
  files: *Prog/Files* 1-3
  functions: *Config/Maint* 3-9
  status: *Config/Maint* 11-8
processor channels, configuring: *Config/Maint* 12-1
processor configuration
  adapter image file: *Config/Maint* 10-11
  communication scan time: *Config/Maint* 10-5
  fault routine: *Config/Maint* 10-6
  I/O status file: *Config/Maint* 10-7
  local rack address: *Config/Maint* 10-5
  main control programs: *Config/Maint* 10-14
  PII: *Config/Maint* 10-12
  start-up procedure: *Config/Maint* 10-3
  STI: *Config/Maint* 10-13
processor configuration screen, PLC–5/30,40,60: *Config/Maint* 10-3
Processor Input Interrupt. *See* PII
processor memory
  comparing: *Prog/Files* 4-1
  files, passwords: *Config/Maint* 14-1
processor memory file, SFC, example ASCII: *Prog/Files* 7-18
processor memory files
  configuring: *Config/Maint* 6-4
  creating offline: *Config/Maint* 6-3
  documenting: *Prog/Ladder* 5-1
    editing keys: *Prog/Ladder* 5-5
    guidelines: *Prog/Ladder* 5-4
  entering symbols: *Prog/Ladder* 5-7
  export: *Prog/Files* 7-6
  importing: *Prog/Files* 7-1
  ladder, example ASCII: *Prog/Files* 7-13
  merging: *Prog/Files* 5-14
  naming: *Prog/Files* 2-4, 2-5
  restoring: *Prog/Files* 3-4
  save and continue: *Prog/Files* 3-1
  save and restore: *Prog/Files* 3-2
  saving: *Prog/Files* 3-1
  using database editor: *Prog/Ladder* 5-8
processor startup procedure: *Config/Maint* 10-3
processor status: *Config/Maint* 11-1, 11-3
  chassis backplane settings: *Config/Maint* 11-10
  extending, PLC–5 VME: *Config/Maint* 11-32
  file addresses: *Config/Maint* 11-1
  runtime date and time: *Config/Maint* 11-9
processor status file, layout: *Config/Maint* 11-1
processor status screen, for configuration, PLC-5/10,12,15,25: *Config/Maint* 10-2
program, ladder, report: *Config/Maint* 19-8
program constant, valid value range: *Instruction* C-1

program constants: *Config/Maint* 8-17
program directory: *Prog/Files* 2-1
  comparing: *Prog/Files* 4-1
  offline: *Config/Maint* 3-8
  online: *Config/Maint* 3-6
program files
  creating: *Prog/Files* 2-1, 2-3
  documenting: *Prog/Ladder* 5-1
    editing keys: *Prog/Ladder* 5-5
    guidelines: *Prog/Ladder* 5-4
  entering symbols: *Prog/Ladder* 5-7
  merging: *Prog/Files* 5-14
  naming: *Prog/Files* 2-4, 2-5
  numbering: *Prog/Files* 2-2; *Prog/SFC* 1-10
  organizing: *Prog/Files* 2-1
  overview: *Prog/Files* 2-2
  privileges: *Config/Maint* 14-11
  undefined: *Prog/SFC* 2-5, 3-13
  using the database editor: *Prog/Ladder* 5-8
program flow
  AFI: *Instruction* 13-14
  JMP and LBL: *Instruction* 13-3
  JSR, SBR, and RET: *Instruction* 13-8
  MCR: *Instruction* 13-2
  OSR: *Instruction* 13-14
  TND: *Instruction* 13-14, 13-17
  UID: *Instruction* 13-17
  UIE: *Instruction* 13-18
program flow instruction
  FOR, BRK, LBL, and RET: *Instruction* 13-5
  OSF: *Instruction* 13-15
  OSR: *Instruction* 13-15
  SFR: *Prog/SFC* 4-16; *Instruction* 13-16
program flow instructions: *Instruction* 13-1
program scan
  main control program: *Config/Maint* 11-14
  overall: *Config/Maint* 11-12
Programming, SDS instruction: *Instruction* 18-2
programming
  DH+ to DH+: *Config/Maint* 5-14
  instructions, operands: *Instruction* C-1
  offline: *Config/Maint* 3-8
  online: *Config/Maint* 3-6
  protecting: *Config/Maint* 14-1
programming software
  exiting: *Config/Maint* 2-6
  function keys: *Config/Maint* 3-4
  input keys: *Config/Maint* 3-3
  installing: *Config/Maint* 1-1, 1-5
  keys: *Config/Maint* 3-3
  main menu: *Config/Maint* 3-1
  navigational keys: *Config/Maint* 3-4
  offline: *Config/Maint* 6-1
  online: *Config/Maint* 5-1
  removing: *Config/Maint* 1-12
  Smart Menu: *Config/Maint* 1-8
  starting
    DH+ Network Menu: *Config/Maint* 2-3
    Smart Menu: *Config/Maint* 2-1
    with an executable: *Config/Maint* 2-3
  user configuration: *Config/Maint* 2-5
  using
    1784-T45: *Config/Maint* 1-2
    6171-IDH: *Config/Maint* 1-2
  verifying installation: *Config/Maint* 1-8
programming terminal
  configuring: *Config/Maint* 4-1
  on DH+: *Config/Maint* 7-5
  online: *Config/Maint* 5-1
  privileges: *Config/Maint* 14-10
  supported by software: *Config/Maint* 1-4
Proportional, Integral, and Derivative instruction: *Instruction* 14-9
protecting programs: *Config/Maint* 14-1

Q

quadrants: *Prog/SFC* 1-14

R

rack
  configuration tables: *Config/Maint* 11-21
  inhibiting: *Config/Maint* 11-21
  resetting: *Config/Maint* 11-21

Config/Maint = Software Configuration and Maintenance
I/OConfig = I/O Configuration Software
Instruction = Instruction Set Reference
Prog/Files = Programming, Files section
Prog/SFC = Programming, SFCs section
Prog/Ladder = Programming, Ladder section

rack addressing: *Config/Maint* 12-29
rack fault, status information: *Config/Maint* 11-23
rack inhibit, status information: *Config/Maint* 11-23
rack reset, status information: *Config/Maint* 11-23
rack size, configuring in remote I/O scanner: *Config/Maint* 12-25, 12-32
racks
  adding: *Config/Maint* 11-25
  removing: *Config/Maint* 11-25
radix, changing: *Config/Maint* 15-2
registration: *Config/Maint* 1-6
relay–type
  IIN: *Instruction* 1-5, 1-6
  OTE: *Instruction* 1-3
  OTL: *Instruction* 1-4
  OTU: *Instruction* 1-5
  XIC: *Instruction* 1-2
  XIO: *Instruction* 1-3
remote I/O, adapter image file: *Config/Maint* 10-11
remote I/O adapter, interpreting status: *Config/Maint* 13-16
remote I/O scanner, interpreting status: *Config/Maint* 13-12
remote programming: *Config/Maint* 5-14
  See also DH+
removing
  passwords: *Config/Maint* 14-3
  software: *Config/Maint* 1-12
renaming, files: *Prog/Files* 5-1
replacing, database editor: *Prog/Ladder* 5-14
reports
  changing title: *Config/Maint* 19-11
  compare: *Prog/Files* 4-4
  configuring printer: *Config/Maint* 19-16
  creating offline: *Config/Maint* 19-5
  creating online: *Config/Maint* 19-3
  file extensions: *Config/Maint* 19-13
  generating: *Config/Maint* 19-1
  options: *Config/Maint* 19-10
  printing: *Config/Maint* 19-13
  types of: *Config/Maint* 19-1, 19-7
  viewing: *Config/Maint* 19-19
requirements, system: *Config/Maint* 1-1
RES instruction: *Instruction* 2-18
Reset instruction: *Instruction* 2-18
resetting racks: *Config/Maint* 13-15, 13-20
resident local rack address, configuring: *Config/Maint* 10-5
restoring
  CAR files: *Prog/Files* 3-7
  files: *Prog/Files* 3-4
RET instruction: *Instruction* 13-8
Retentive Timer On instruction: *Instruction* 2-8
Return instruction: *Instruction* 13-8
RS232C, RS422, RS423, configuration: *Config/Maint* 12-5
RTO instruction: *Instruction* 2-8
run times, determining: *Instruction* B-12
rung
  creating: *Prog/Ladder* 1-3
  definition: *Prog/Ladder* 1-1
  entering: *Prog/Ladder* 3-1
  example format: *Prog/Ladder* 1-2
  logic, writing: *Prog/Ladder* 1-3
  moving: *Prog/Ladder* 2-7
  selecting: *Prog/Ladder* 2-6
  single vs. multiple: *Prog/Ladder* 2-4

S

save and continue: *Prog/Files* 3-1
saving, processor memory: *Prog/Files* 3-2
saving files: *Prog/Files* 3-1
SBR instruction: *Instruction* 13-8
scaling, to engineering units: *Instruction* 14-4
scan list
  creating
    expanded local I/O: *Config/Maint* 12-32
    remote I/O: *Config/Maint* 12-25, 12-32
  expanded local I/O: *Config/Maint* 13-19
  extended local I/O: *Config/Maint* 12-30

limitations: *Config/Maint* 12-25, 12-32
modifying: *Config/Maint* 11-25
expanded local I/O: *Config/Maint* 12-32
remote I/O: *Config/Maint* 12-25
remote I/O: *Config/Maint* 13-14
scan sequence, SFC: *Instruction* B-6
scan time
convergent: *Instruction* B-13
divergent: *Instruction* B-13
steady-state: *Instruction* B-13
scanner, monitoring status: *Config/Maint* 13-12
scanner mode, configuring: *Instruction* 15-12, 15-13
scanner, expanded local I/O
creating a scan list: *Config/Maint* 12-32
modifying a scan list: *Config/Maint* 12-25, 12-32
scanner, remote I/O
configuring channels 1A-2B: *Config/Maint* 12-23
creating a scan list: *Config/Maint* 12-25, 12-32
SDS CAR: *Instruction* 18-2
SDS instruction: *Instruction* 18-1
searching
assisted: *Prog/Ladder* 4-1
configuring: *Prog/Ladder* 2-4, 4-13
database editor: *Prog/Ladder* 5-12
replacing: *Prog/Ladder* 5-14
edit zones: *Prog/Ladder* 4-10
instructions/addresses: *Prog/Ladder* 4-4
ladder editor: *Prog/Ladder* 4-1
macros: *Prog/SFC* 4-15
replacing: *Prog/Ladder* 4-8
SFC: *Prog/SFC* 5-3
standard: *Prog/Ladder* 4-3
types: *Prog/Ladder* 4-4
undocumented instructions: *Prog/Ladder* 4-12
searching and replacing, SFC: *Prog/SFC* 5-5
Selectable Timed Interrupt. *See* STI
selecting
offline files: *Config/Maint* 6-4
rungs: *Prog/Ladder* 2-6
selection branch: *Prog/SFC* 1-3
editing: *Prog/SFC* 4-1
entering: *Prog/SFC* 4-4, 4-6
scanning sequence: *Instruction* B-6
sequencer
applying: *Instruction* 12-1
instructions: *Instruction* 12-1
SQI: *Instruction* 12-2
SQL: *Instruction* 12-2
SQO: *Instruction* 12-2
Sequencer Input instruction: *Instruction* 12-2
Sequencer Load instruction: *Instruction* 12-2
Sequencer Output instruction: *Instruction* 12-2
sequential function chart
branches: *Prog/SFC* 4-1
building: *Prog/SFC* 2-4, 3-11
constraints: *Instruction* B-4
creating macros: *Prog/SFC* 4-12, 4-13
display, configuring: *Prog/SFC* 1-12
editing: *Prog/SFC* 5-1
editor: *Prog/SFC* 2-1, 3-1, 5-1
forcing transitions: *Config/Maint* 17-6; *Prog/SFC* 5-7
GOTOs and labels: *Prog/SFC* 4-7
macros, searching: *Prog/SFC* 4-15
memory requirements: *Instruction* B-3
quadrants: *Prog/SFC* 1-14
reference information: *Instruction* B-1
scanning sequence: *Instruction* B-6
searching: *Prog/SFC* 5-3
searching and replacing: *Prog/SFC* 5-5
status information: *Instruction* B-1
using subcharts: *Prog/SFC* 4-15
zooming: *Prog/SFC* 5-6
Sequential Function Chart Reset instruction: *Prog/SFC* 4-16; *Instruction* 13-16
serial port
communications: *Config/Maint* 5-15, 5-18
configuration: *Config/Maint* 12-5
configuring: *Config/Maint* 5-15

ASCII (user mode): *Config/Maint* 12-15
communication mode change: *Config/Maint* 12-18
DF1 master: *Config/Maint* 12-11
DF1 point-to-point: *Config/Maint* 12-6
DF1 slave: *Config/Maint* 12-8
DF1 master status: *Config/Maint* 13-5
DF1 point-to-point status: *Config/Maint* 13-4
DF1 slave status: *Config/Maint* 13-5
modem: *Config/Maint* 5-19
monitoring: *Config/Maint* 13-4
protocols: *Config/Maint* 12-5
user mode (ASCII): *Config/Maint* 13-7
SFC
assigning step/tran names: *Prog/SFC* 3-8
building: *Prog/SFC* 2-3, 2-4, 3-3, 3-11
building blocks: *Prog/SFC* 1-1
configuration information: *Prog/SFC* 5-10
constraints: *Instruction* B-4
conversion: *Prog/SFC* 5-1
creating: *Prog/SFC* 2-1, 3-1
structures: *Prog/SFC* 4-1
drawing: *Prog/SFC* 1-1, 1-7
edit mode: *Prog/SFC* 1-14
editing: *Prog/SFC* 5-1
entering
GO TO/label: *Prog/SFC* 4-11, 4-12
selection branch: *Prog/SFC* 4-4, 4-6
simultaneous branch: *Prog/SFC* 4-5, 4-7
step actions, transitions: *Prog/SFC* 3-3
step/transition pairs: *Prog/SFC* 2-3
entering comments: *Prog/SFC* 3-9
example
scanning sequence: *Instruction* B-9
standard building blocks: *Prog/SFC* 1-7
example ASCII export file: *Prog/Files* 7-18
forcing transitions: *Config/Maint* 17-6; *Prog/SFC* 5-7
getting started: *Prog/SFC* 1-1
GOTOs and labels: *Prog/SFC* 1-8
macros: *Prog/SFC* 4-12
assigning comments: *Prog/SFC* 4-14
creating: *Prog/SFC* 4-13
deleting step/tran: *Prog/SFC* 4-14
searching: *Prog/SFC* 4-15, 5-3
searching within: *Prog/SFC* 4-15
memory requirements: *Instruction* B-3
monitor mode: *Prog/SFC* 1-14
path: *Prog/SFC* 1-2
postscan configuration: *Prog/SFC* 5-8
programming offline: *Prog/SFC* 2-3, 3-3
quadrants: *Prog/SFC* 1-14
reference information: *Instruction* B-1
reports: *Config/Maint* 19-7
scan configuration: *Prog/SFC* 5-8
scan sequences, considerations: *Prog/SFC* 1-16
scanning sequence
example: *Instruction* B-9
selection branch: *Instruction* B-6
simultaneous branch: *Instruction* B-8
scanning sequences, step/transition: *Instruction* B-6
searching: *Prog/SFC* 5-3
searching and replacing: *Prog/SFC* 5-5
selection branch: *Prog/SFC* 1-3
simultaneous branch: *Prog/SFC* 1-4, 1-5
special structures: *Prog/SFC* 1-16
status information: *Prog/SFC* 5-8; *Instruction* B-1
step: *Prog/SFC* 1-1
step actions: *Prog/SFC* 1-2
subcharts: *Prog/SFC* 4-15
timer number: *Prog/SFC* 5-11
transition: *Prog/SFC* 1-2
zooming: *Prog/SFC* 5-6
SFC editor, program mode: *Prog/SFC* 5-3
SFR instruction: *Prog/SFC* 4-16; *Instruction* 13-16
shift register instruction
applying: *Instruction* 11-1
BSL and BSR: *Instruction* 11-1
FFL and FFU: *Instruction* 11-5

LFL and LFU: *Instruction* 11-5
short cuts: *Config/Maint* 3-4
simultaneous branch: *Prog/SFC* 1-4
editing: *Prog/SFC* 4-1
entering: *Prog/SFC* 4-5, 4-7
post scan mode: *Prog/SFC* 1-5
scanning sequence: *Instruction* B-8
single rung display: *Prog/Ladder* 2-4
Smart Directed Sequencer (SDS) Instruction
entering: *Instruction* 18-3
overview: *Instruction* 18-1
programming: *Instruction* 18-2
Smart Directed Sequencer Instruction: *Instruction* 18-1
Smart Menu
during software installation: *Config/Maint* 1-7
fields: *Config/Maint* 1-10
installing: *Config/Maint* 1-8
modifying: *Config/Maint* 1-9
specifying
6200 software: *Config/Maint* 1-11
other software: *Config/Maint* 1-11
starting: *Config/Maint* 2-2
starting the software: *Config/Maint* 2-1
using: *Config/Maint* 2-1
snow control. *See* Programming Terminal, configuring
software
exiting: *Config/Maint* 2-6
function keys: *Config/Maint* 3-4
input keys: *Config/Maint* 3-3
installing: *Config/Maint* 1-1
keys: *Config/Maint* 3-3
main menu: *Config/Maint* 3-1
navigational keys: *Config/Maint* 3-4
offline: *Config/Maint* 6-1
online: *Config/Maint* 5-1
registration: *Config/Maint* 1-8
removing: *Config/Maint* 1-12
starting
DH+ Network Menu: *Config/Maint* 2-3
Smart Menu: *Config/Maint* 2-1
with an executable: *Config/Maint* 2-3
SQI instruction: *Instruction* 12-2
SQL instruction: *Instruction* 12-2
SQO instruction: *Instruction* 12-2
SQR instruction: *Instruction* 4-19
Square Root instruction, SQR: *Instruction* 4-19
standard searching: *Prog/Ladder* 4-3
starting
clearing memory: *Config/Maint* 3-11
DH+ Network Menu: *Config/Maint* 2-3
Smart Menu: *Config/Maint* 2-1
software: *Config/Maint* 2-1
starting I/O group: *Config/Maint* 11-23
startup routines: *Config/Maint* 10-4
station, privileges: *Config/Maint* 14-10
status
file: *Config/Maint* 8-6
Local I/O: *Config/Maint* 13-19
local I/O rack faults: *Config/Maint* 13-20
monitoring the processor: *Config/Maint* 11-1
of I/O racks: *Config/Maint* 13-14
processor, layout: *Config/Maint* 11-1
processor channels: *Config/Maint* 13-1
rack faults: *Config/Maint* 13-15
rack inhibit: *Config/Maint* 13-15, 13-20
rack re-scans: *Config/Maint* 13-15, 13-20
resetting racks: *Config/Maint* 13-15, 13-20
status bits
global control: *Config/Maint* 11-20
global status: *Config/Maint* 11-18
status data, serial port
DF1 master: *Config/Maint* 13-5
DF1 point-to-point: *Config/Maint* 13-4
DF1 slave: *Config/Maint* 13-5
user mode (ASCII): *Config/Maint* 13-7
status file, size: *Config/Maint* 8-5
status information
active node table: *Config/Maint* 11-12
arithmetic flags: *Config/Maint* 11-11
chassis addressing: *Config/Maint* 11-10
EEPROM status: *Config/Maint* 11-10

Config/Maint = Software Configuration and Maintenance
I/OConfig = I/O Configuration Software
Instruction = Instruction Set Reference
Prog/Files = Programming, Files section
Prog/SFC = Programming, SFCs section
Prog/Ladder = Programming, Ladder section

extending, PLC–5 VME: *Config/Maint* 11-32
I/O: *Config/Maint* 11-15
indexed addressing offset: *Config/Maint* 11-11
main control program scan: *Config/Maint* 11-14
PIIs: *Config/Maint* 11-14
processor: *Config/Maint* 11-8, 11-9
See also processor status
processor memory protection: *Config/Maint* 11-10
processor station address: *Config/Maint* 11-12
program scan: *Config/Maint* 11-12
RAM backup: *Config/Maint* 11-10
SFC: *Instruction* B-1
STIs: *Config/Maint* 11-15
status messages, ladder editor: *Prog/Ladder* 2-8
steady-state, scan time: *Instruction* B-13
step
actions: *Prog/SFC* 1-2, 2-1, 3-1
display configuration: *Prog/SFC* 1-12
entering: *Prog/SFC* 3-4
qualifiers: *Prog/SFC* 3-4, 3-6
reference numbers: *Prog/SFC* 1-12
assigning names: *Prog/SFC* 3-8
building block: *Prog/SFC* 1-1
default file: *Prog/Ladder* 2-2
entering comments: *Prog/SFC* 3-9
monitoring: *Prog/SFC* 5-10
scanning sequence: *Instruction* B-6
step/transition pairs: *Prog/SFC* 2-3
STI
configuration fields: *Config/Maint* 10-13
configuring: *Config/Maint* 10-13
STIs, status: *Config/Maint* 11-15
string data, monitoring: *Config/Maint* 15-11
SUB instruction: *Instruction* 4-24
sub-member, data storage: *Config/Maint* 8-3
subcharts: *Prog/SFC* 4-15
Subroutine Header instruction: *Instruction* 13-8
Subtract instruction, SUB: *Instruction* 4-24
symbol, address: *Config/Maint* 8-15
symbols
import/export. *See* importing, documentation
ladder editor display: *Prog/Ladder* 2-4
report: *Config/Maint* 19-9
syntax errors, responding: *Prog/Files* 6-7
system mode, status: *Config/Maint* 13-4
system requirements: *Config/Maint* 1-1

T

Temporary End instruction: *Instruction* 13-14, 13-17, 13-18
terminal. *See* programming terminal
Test Buffer For Line instruction: *Instruction* 17-3
testing, edits: *Prog/Ladder* 4-20
modes: *Prog/Ladder* 4-21
timer
accuracy: *Instruction* 2-3
instruction parameters: *Instruction* 2-1, 2-11
RES: *Instruction* 2-18
RTO: *Instruction* 2-8
TOF: *Instruction* 2-6
TON instruction: *Instruction* 2-3
Timer Off Delay instruction: *Instruction* 2-6
Timer On Delay instruction: *Instruction* 2-3
timers: *Instruction* 2-1
timing
block transfer: *Instruction* 15-12, 15-13
instructions: *Instruction* A-1
title, report: *Config/Maint* 19-11
TND instruction: *Instruction* 13-14, 13-17, 13-18
TOD instruction: *Instruction* 6-2
TOF instruction: *Instruction* 2-6
TON instruction: *Instruction* 2-3
transition
assigning names: *Prog/SFC* 3-8
building block: *Prog/SFC* 1-2
entering comments: *Prog/SFC* 3-9

forcing: *Config/Maint* 17-6; *Prog/SFC* 5-7
scanning sequence: *Instruction* B-6
type, data storage: *Config/Maint* 8-2

U

understanding, files: *Prog/Files* 1-1
units, engineering, scaling: *Instruction* 14-4
unused, channel configuration: *Config/Maint* 12-28
user configuration files: *Config/Maint* 2-5
user control bits
local rack address: *Config/Maint* 10-5
start-up procedure: *Config/Maint* 10-3
User Interrupt Disable, UID: *Instruction* 13-17
User Interrupt Enable, UIE: *Instruction* 13-18
using
keywords: *Prog/Files* 6-6
undefined program files: *Prog/SFC* 2-5, 3-13
utilities, files: *Prog/Files* 5-1

V

viewing, reports: *Config/Maint* 19-19

W

WHO: *Config/Maint* 2-4
See also DH+ Network Menu
active: *Config/Maint* 7-5
listen: *Config/Maint* 7-3
station diagnostics
displaying: *Config/Maint* 7-6
PLC-2: *Config/Maint* 7-7
PLC-3: *Config/Maint* 7-8
PLC-5: *Config/Maint* 7-9
PLC-5/250: *Config/Maint* 7-11
PLC-5/40, -5/60: *Config/Maint* 7-10
using: *Config/Maint* 7-2
words, data storage: *Config/Maint* 8-2
writing
branch logic: *Prog/Ladder* 1-5
rung logic: *Prog/Ladder* 1-3
to an EEPROM: *Prog/Files* 3-8

X

.X5 file format: *Prog/Files* B-1
XIC instruction: *Instruction* 1-2
XIO instruction: *Instruction* 1-3
XOR instruction: *Instruction* 5-5

Z

zooming: *Prog/SFC* 5-6

© Allen-Bradley Comapny, Inc.
PLC is a registered trademark of Allen-Bradley Company, Inc.
PLC-5 and Pyramid Integrator are trademarks of Allen-Bradley Company, Inc.
DH+ is a trademarks of Allen-Bradley Company, Inc.
IBM is a registered trademark of International Business Machines Corporation

Customer Support

If you need additional assistance on using your software, Allen-Bradley offers telephone and on-site product support at Customer Support Centers worldwide.

For technical assistance on the telephone, first contact your local sales office, distributor, or system integrator. If additional assistance is needed, then contact your local Customer Support Center or contact System Support Services.

In the United States and Canada

If you have a SupportPlus agreement or your software is under warranty, you can contact System Support Services at: 1-800-289-2279. Have your support contract or software registration number available.

For assistance that requires on-site support, contact your local sales office, distributor, or system integrator. During non-office hours, contact the Allen-Bradley 24-hour Hot Line at 1-800-422-4913.

Outside of the United States

Contact your local Customer Support Center at:

Region or Area	Customer Support Center Telephone Number
Canada (Cambridge, Ontario)	519-623-1810
Latin America (Milwaukee)	414-382-2000
United Kingdom (Milton Keynes)	44-908 838800
Europe (Amsterdam)	31-2975 43500
France (Paris)	(33-1) 4778 1402
Germany (Gruiten)	(49) 2104 6900
Italy (Milan)	(39-2) 4830 0381
Asia Pacific (Hong Kong)	(852) 873-1342

For assistance that requires on-site support, contact your local office, distributor, or system integrator. During non-office hours, contact your local Customer Support Center.

ASCII Character Codes

Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex
[ctrl-@] NUL	0	00	SPACE	32	20	@	64	40	`	96	60
[ctrl-A] SOH	1	01	!	33	21	A	65	41	a	97	61
[ctrl-B] STX	2	02	"	34	22	B	66	42	c	99	63
[ctrl-C] ETX	3	03	#	35	23	C	67	43	b	98	62
[ctrl-D] EOT	4	04	$	36	24	D	68	44	d	100	64
[ctrl-E] ENQ	5	05	%	37	25	E	69	45	e	101	65
[ctrl-F] ACK	6	06	&	38	26	F	70	56	f	102	66
[ctrl-G] BEL	7	07	'	39	27	G	71	47	g	103	67
[ctrl-H] BS	8	08	(	40	28	H	72	48	h	104	68
[ctrl-I] HT	9	09	)	41	29	I	73	49	i	105	69
[ctrl-J] LF	10	0A	*	42	2A	J	74	4A	j	106	6A
[ctrl-K] VT	11	0B	+	43	2B	K	75	4B	k	107	6B
[ctrl-L] FF	12	0C	,	44	2C	L	76	4C	l	108	6C
[ctrl-M] CR	13	0D	-	45	2D	M	77	4D	m	109	6D
[ctrl-N] SO	14	0E	.	46	2E	N	78	4E	n	110	6E
[ctrl-O] SI	15	0F	/	47	2F	O	79	4F	o	111	6F
[ctrl-P] DLE	16	10	0	48	30	P	80	50	p	112	70
[ctrl-Q] DC1	17	11	1	49	31	Q	81	51	q	113	71
[ctrl-R] DC2	18	12	2	50	32	R	82	52	r	114	72
[ctrl-S] DC3	19	13	3	51	33	S	83	53	s	115	73
[ctrl-T] DC4	20	14	4	52	34	T	84	54	t	116	74
[ctrl-U] NAK	21	15	5	53	35	U	85	55	u	117	75
[ctrl-V] SYN	22	16	6	54	36	V	86	56	v	118	76
[ctrl-W] ETB	23	17	7	55	37	W	87	57	w	119	77
[ctrl-X] CAN	24	18	8	56	38	X	88	58	x	120	78
[ctrl-Y] EM	25	19	9	57	39	Y	89	59	y	121	79
[ctrl-Z] SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ctrl-[ESC	27	1B	;	59	3B	[	91	5B	{	123	7B
[ctrl-\] FS	28	1C	<	60	3C	\	92	5C	|	124	7C
ctrl-] GS	29	1D	=	61	3D	]	93	5D	}	125	7D
[ctrl-^] RS	30	1E	>	62	3E	^	94	5E	~	126	7E
[ctrl-_] US	31	1F	?	63	3F	SPACE	95	5F	DEL	127	7F

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

As a subsidiary of Rockwell International, one of the world's largest technology companies — Allen-Bradley meets today's challenges of industrial automation with over 85 years of practical plant-floor experience. More than 13,000 employees throughout the world design, manufacture and apply a wide range of control and automation products and supporting services to help our customers continuously improve quality, productivity and time to market. These products and services not only control individual machines but integrate the manufacturing process, while providing access to vital plant floor data that can be used to support decision-making throughout the enterprise.

With offices in major cities worldwide

WORLD HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (414) 382-2000
Telex: 43 11 016
FAX: (414) 382-4444

EUROPE/MIDDLE EAST/AFRICA HEADQUARTERS
Allen-Bradley Europa B.V.
Amsterdamseweg 15
1422 AC Uithoorn
The Netherlands
Tel: (31) 2975/60611
Telex: (844) 18042
FAX: (31) 2975/60222

ASIA/PACIFIC HEADQUARTERS
Allen-Bradley (Hong Kong) Limited
Room 1006, Block B, Sea View Estate
28 Watson Road
Hong Kong
Tel: (852) 887-4788
Telex: (780) 64347
FAX: (852) 510-9436

CANADA HEADQUARTERS
Allen-Bradley Canada Limited
135 Dundas Street
Cambridge, Ontario N1R 5X1
Canada
Tel: (519) 623-1810
FAX: (519) 623-8930

LATIN AMERICA HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (414) 382-2000
Telex: 43 11 016
FAX: (414) 382-2400

Publication 6200-6.4.11 – April 1992
Supersedes Publication 6200-6.4.11 – November 1991

PN 955111-41
Printed in USA